TREE STRUCTURED ALGORITHMS FOR SCHEDULING ACTIVITIES AND RESOURCES IN A CONTINUUM OF TIME

By

MARTIN JAMES WERTHEIM

Bachelor of Science

Duke University

Durham, North Carolina

1969

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
MASTER OF SCIENCE
July, 1973

TREE STRUCTURED ALGORITHMS FOR SCHEDULING

ACTIVITIES AND RESOURCES IN A

CONTINUUM OF TIME

Thesis Approved:

James R. Van Doren
Thesis Adviser

G. E. Hedrick

D. D. Durham
Dean of the Graduate College

PREFACE

This thesis is concerned with the development of a computer program to solve a particular class of scheduling problems. The primary objective is to implement tree structured searching techniques in the search for a schedule.

I wish to express my thanks to my thesis adviser, Dr. James R. Van Doren, for suggesting the topic of this thesis and providing invaluable assistance and guidance. Thanks are also due to other faculty members of the Department of Computing and Information Sciences, for their helpful advice and suggestions. A special note of thanks is due to Dr. Donald W. Grace who pointed out that one aspect of resource assignment based on attributes was a special case of the transportation problem.

Finally, I wish to thank the citizens of the City of Stillwater and the State of Oklahoma for providing the environment which helped make my education at Oklahoma State University a truly remarkable experience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

A_i	i'th attribute group
a_i	starting time of the i'th window
b_i	ending time of the i'th window
c_i	start of actual scheduled time within i'th window
d_i	end of actual scheduled time within i'th window
Δ_i	actual time required by i'th activity
q_j	number of units of j'th resource class
$R(A_i)$	number of units required of attribute group A_i.
x_i	activity
y_j	resource unit or resource class

CHAPTER I

INTRODUCTION

A schedule can be defined as a time plan, or a list of times, for the occurrence of a group of events or procedures. The problems incurred in creating schedules vary greatly from one application to another; however, there is one common characteristic inherent in all scheduling problems, the need to make decisions. This decision making requirement usually arises due to some limitations of time or resources. Often a choice must be made between two or more possible schedules as to which schedule is, in some sense, optimal.

Van Doren (1) has observed that scheduling problems take on the characteristics of a three dimensional constrained search. The three dimensions are activities, resources, and time. The following examples, taken from industrial scheduling and space flight scheduling, illustrate the three dimensional nature of these problems.

Muth and Thompson (2) have defined industrial scheduling as a problem of making decisions on how to use each manufacturing facility at each instant of time, taking into account such considerations as availability of resources, cost of implementing decisions, due dates, and so forth. They have identified three major classes of industrial scheduling problems. In the first of these, the job-shop problem, a firm contains one or more work centers, and each unit of product manufactured must pass through each work center at some stage of the manufacturing process. The production of each unit is an activity, and the work centers, composed of machines and workers, are resources. The goal of a job shop schedule might be to meet a production deadline (time), or to minimize the total time required to complete all jobs. A typical constraint might be that a work center can operate on, at most, one product at any instant of time. A second class of problems arises when a firm keeps an inventory of goods and must decide periodically when and how many goods to manufacture. In making these decisions, the firm must take into account constraints on the availability of resources such as raw material, labor, and capital. A third class of problems, single project scheduling, arises when a project consisting of several distinct tasks (activities) must be completed by a certain due date (time constraint). In addition to constraints imposed by resource limitations, constraints may arise due to requirements that some tasks be performed either before or after others.

Another example which illustrates the three dimensional nature of the problem can be found in the scheduling problems associated with NASA's space shuttle program (1). Activities to be scheduled include shuttle flights, maintenance of orbiters, and deliveries of payloads to a given orbit. Resources to be scheduled include orbiters, solid rocket boosters, flight crews, etc. The time dimension may involve several windows of time, that is, intervals of time during which an activity must take place.

In many cases, more than one solution can be found for a particular scheduling problem. In such cases it may be desirable to find all feasible solutions and choose from among the feasible solutions one solution which is optimal. The problem, then, may be compared to linear programming problems in which it is desired to maximize or minimize an objective function subject to various constraints.

Because of the great variety of scheduling problems, it is highly unlikely that a computer program could be developed that would be general enough to handle all types of scheduling problems. Indeed, most programs that have been written are designed to solve one particular problem. However, programs can be developed with enough generality so that certain classes of problems with common characteristics and requirements could be solved. The subject of this report is the development of a computer program to solve scheduling problems of one particular class.

In the class of problems investigated in this report, an activity is a non-recurring event that extends over a continuous time interval and requires the use of one or more resources. A resource class is a collection of one or more identical resource units. A window of time is a time interval during which an activity must be scheduled. There are m activities to be scheduled and n classes of resource units to be allocated. For simplicity, the restriction is made that an activity may require at most one unit of each resource class. Associated with each activity are one or more windows of time, and a duration time which is the total time necessary to complete an activity. The problem is to find an actual starting and ending time for each activity such that each activity is scheduled within one of the windows of time associated with that activity, and that each resource unit is assigned to at most one activity at any one instant of time. In an extension of this problem, one or more attributes are associated with each resource class, thus forming attribute groups. Each attribute group consists of one or more resource classes and each resource class may belong to one or more attribute groups. Activity requirements are stated in terms of attribute groups rather than resource classes, that is to say, each activity requires exactly one unit of one or more attribute groups.

Previous work in this field includes investigations of problems of a similar nature. Bratley, et. al. (3), have investigated the problem of scheduling n tasks on a single resource. Each task has a specified earliest start time, latest completion time and number of time units required. They have developed an algorithm to find a schedule which minimizes the total elapsed time to complete all jobs. The approach they have taken is to consider all possible orderings of n tasks on a single resource. Davis and Heidorn (4) have investigated the problem of scheduling multiple projects requiring multiple resources, using techniques originally developed to solve line balancing problems. Their goal also was to minimize project duration. In each of the investigations attempts were made to force a discrete resolution on the time dimension. For example, Davis and Heidorn (4) consider a task requiring n units of time as n separate tasks each of which requires one unit of time. However, as Van Doren (1) has pointed out, it may be highly desirable to treat the time dimension as a continuum. One reason for this is that a discrete time resolution may lead to methods of scheduling in which each unit of time is examined, which would magnify the combinatorial complexity of the problem. Another reason is that, in some problems, the times required and the windows of time for different activities would vary greatly in magnitude. In such cases it would be difficult to decide on the proper size of a time unit.

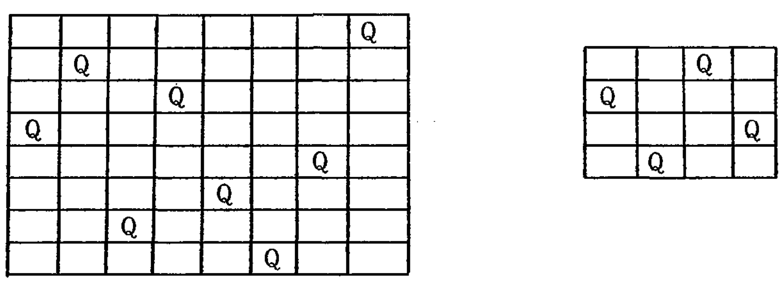
It should be emphasized that the major goal of this investigation has been the examination of methods used in searching for a schedule. Therefore, the goal that has been adopted is the determination of whether a schedule exists rather than the detection of a schedule that is optimal. When appropriate, however, various criteria of optimality will be mentioned, along with suggestions to achieve these criteria.

The search methods used to find a schedule are based on the concepts of decision trees and backtrack programming as presented by Golomb and Baumert (5). These concepts are outlined in Chapter II. It was decided that the investigation should proceed in a stepwise manner, beginning with the solution of some simple problems and then progressing in successive steps of enlargement and refinement in solving more complex problems, until the class of problems discussed earlier could be attacked in its full generality. Thus, the first step in the investigation was the application of decision trees and backtrack programming to the solution of a fairly well-known problem, the eight queens problem of chess. The reasons for this step are that the problem is well defined and that it has certain similarities to the scheduling problems investigated in this report. Two programs which are described in Chapter II, were written to solve the eight queens problem. Chapter III describes a program written to solve a fairly simple scheduling problem, namely scheduling a single resource unit. Chapter IV describes an enlargement of this program to schedule a single class of resource units. Chapters V and VI describe a program to solve a more complex problem, namely scheduling multiple classes of resource units, and, finally, Chapter VII describes the ultimate goal of the investigation, scheduling multiple resource classes, where selection is based on attribute groups. Suggestions for further work are outlined in Chapter VIII.

CHAPTER II

THE EIGHT QUEENS PROBLEM

To gain insight into possible search techniques which would be useful in a scheduling program, it was decided to begin the investigation by writing two programs to find solutions to the eight queens chessboard problem. The problem is to place eight queens on a chessboard in such a way that no queen may be attacked by another queen. A queen is safe from attack if no other queen is positioned on the same row, the same column or the same diagonal. Solutions to this problem are well known. A generalization of the problem is to place n queens on an n x n chessboard. Figure 1 shows one solution to the eight queens problem and one solution to the four queens problem.



Eight Queens　　　　Four Queens

Figure 1. Solutions to Eight Queens and Four Queens Problems.

A partial analogy can be drawn between the eight queens problem and the problem of scheduling a single resource unit. Consider the entire chessboard as a unit of resource, the rows of the chessboard as periods of time, and the columns of the chessboard as activities, each of which requires exactly one period of time. In this analogy the three dimensional view reduces to two dimensions because there is only one resource unit. There are three constraints on the problem two of which have a direct analogy with a realistic scheduling problem. The constraint that not more than one queen may occupy a particular row is analogous to the restriction that the resource unit may be allocated to only one activity during a given time period. The restriction that not more than one queen may occupy a column corresponds to the fact that each activity requires the resource during exactly one time period. The third constraint of course concerns avoiding diagonal placement.

A brute force approach to the problem would be to examine each combination of eight squares on a 64 square chessboard. There are $\binom{64}{8}$ or 4,426,165,368 combinations to be examined. However, it can be observed immediately that each column must be occupied by exactly one queen. The problem then reduces to a search of each column for a possible square to be occupied. The squares must be chosen so that no two queens occupy the same row or the same diagonal. The problem can be represented by a tree structure in which each level of the tree corresponds to a column and each node corresponds to a square within that column. The root of the tree is a dummy node and is considered to be at level zero. Figure 2 shows the tree structure corresponding to the four queens problem. Each path from the root of the tree to a leaf corresponds to a choice of one square for every column; for example, the leftmost path of the tree corresponds to the placement of a queen in the first square of each column.

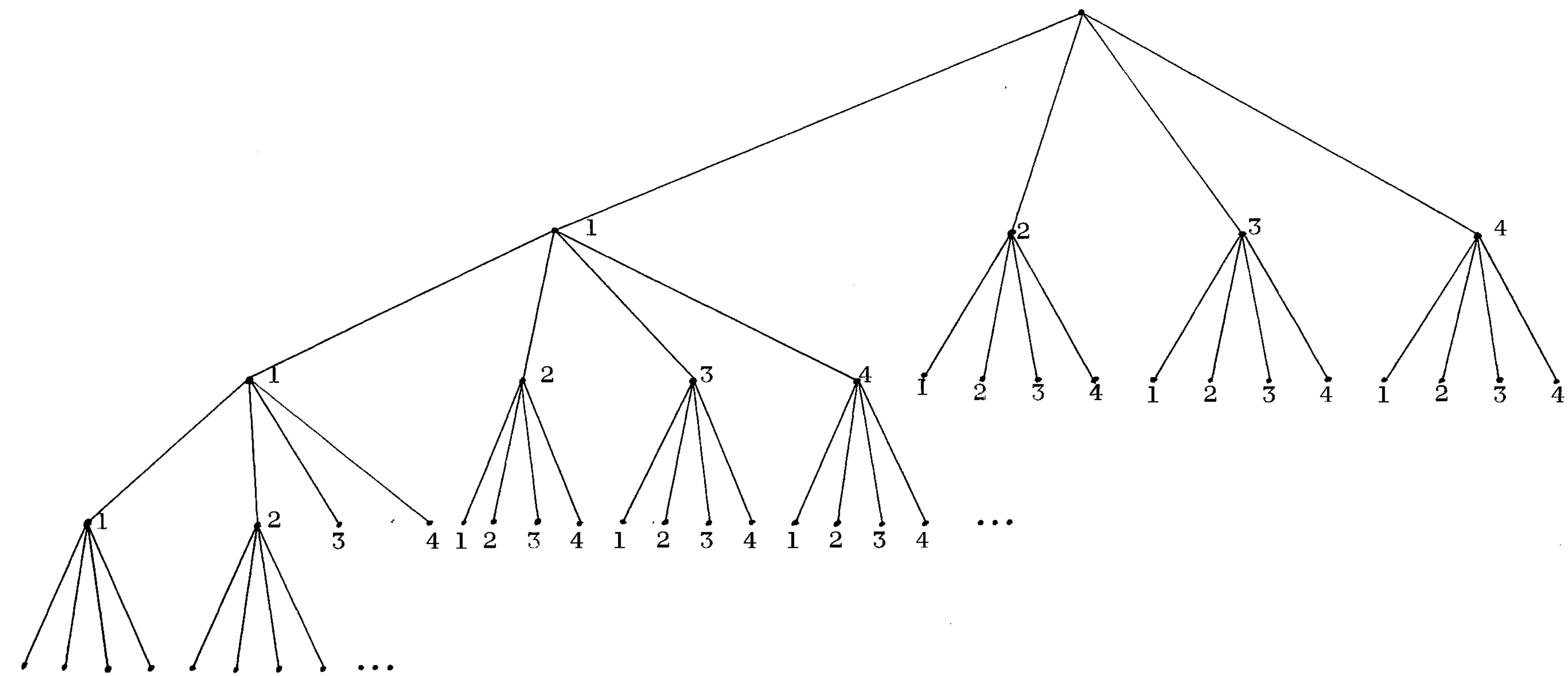


Figure 2. Tree Structure Corresponding to Four Queens Problem.

There are 256 leaves in the tree; therefore, one might suppose that there are 256 alternatives to be examined. However, a closer examination of the tree structured nature of the problem reveals that the number of alternatives to be examined can be reduced. Consider again the left-most path of the tree. Traversing the arc from the root of the tree to its left-most son corresponds to placing a queen on the first square of column one. Traversing the arc from this node to its left-most son corresponds to placing a queen on the first square of column two. Since no solution to the problem can contain two queens in the same row, a conflict condition (constraint violation) exists. Furthermore, it is not necessary to examine any nodes beneath the left-most node at level two; in effect, the tree may be pruned at this node.

Whenever a conflict condition is detected, the right brother of the current node is examined, that is to say, an attempt is made to place a queen on the next square of the column currently being examined. Placing a queen on the second square of column two would also result in a conflict condition since two queens would occupy the same diagonal. However, placing a queen in the third square of column two would cause no conflict. When the examination of a node does not result in a conflict condition, the sons of that node are examined, that is to say, an examination of column three is begun by attempting to place a queen on square one of column three. It turns out that, in the four queens problem with queens placed in column one, square one, and column two, square three, placing a queen anywhere in column three will cause a conflict condition. When all alternatives at a given level result in a conflict condition, then the decision process backtracks one level; in this case it returns to column two and examines the next alternative, namely, placing a queen on square number four of column two. The first nine board configurations to be examined are shown in Figure 3.

When a leaf of the tree is examined and no conflict condition is detected, then the path from the root of the tree to the leaf corresponds to a solution. If only one solution to the problem is desired, then the solution can be reported and the procedure terminated at this point. If all solutions are desired, then the solution can be reported and the search continued by examining the next leaf. If no solution exists, or if the attempt is made to find all solutions, the search terminates after the right-most node of level one (and all of its sons) have been examined.

The method of tree searching described by the example in the preceding paragraphs is known as a depth-first tree search. It should be noted that no explicit data structure corresponding to a tree need be constructed. The tree structure is inherent in the decision making process.

Another method of traversing decision trees is the breadth-first approach. With this method, all nodes of a given level are examined in one step, thus producing the effect of traversing all paths of the tree in parallel. An actual tree structure is constructed so that parallel processing of decision paths can be simulated. One method of construction is to use a binary tree to represent the decision tree under consideration (6). Each node of the binary tree has the representation shown in Figure 4. The left link of each node points to the left son of that node, and the right link of each node points to the brother on the immediate right if one exists, otherwise, the right link

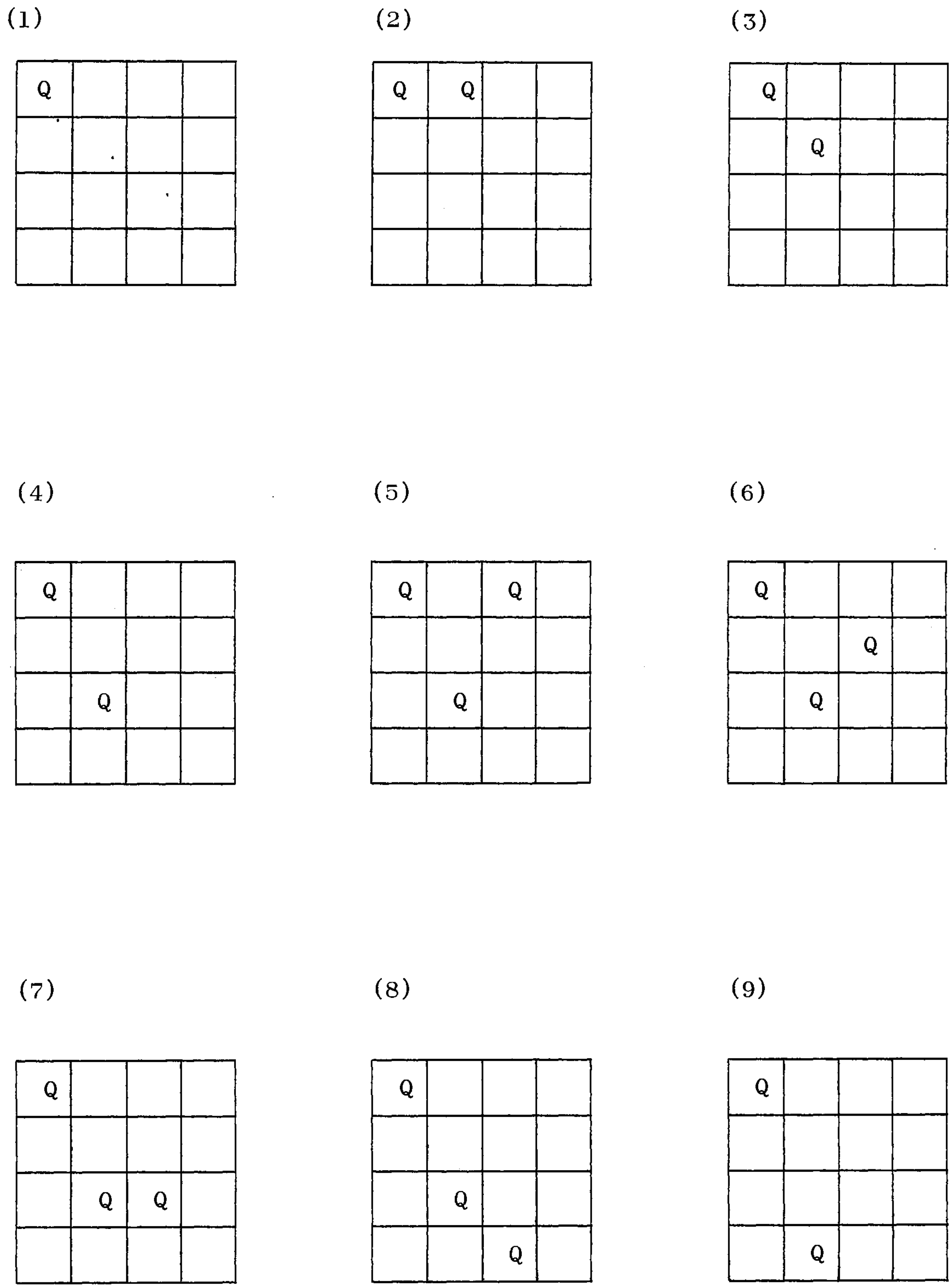


Figure 3. First Nine Board Configurations to be Examined in Four Queens Problem.

is used as a thread and points to the father. An example of a tree and its binary representation is shown in Figure 5.

Left Link	Information	Right Link

Figure 4. One Node of a Binary Tree.

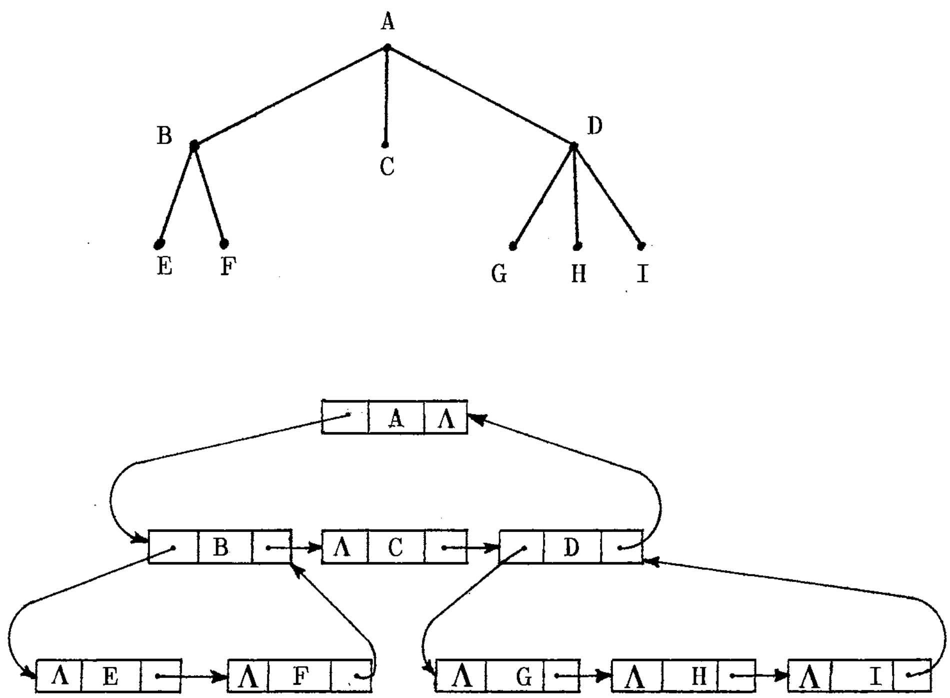


Figure 5. A Tree and Its Binary Representation.

A linked list of available storage is required, along with routines to allocate nodes from the available list and to return nodes which are no longer needed to the available list. The tree is constructed as a binary tree. Processing a level of the tree consists of examining each node of the previous level and for each node of the previous level, determining which alternatives at the current level do not cause a conflict condition. All conflict free alternatives are attached as sons of the node being examined. If no conflict free alternatives are found, then the node being examined may be removed from the tree and returned to the available list. If a node is pruned which has no brothers, then the father of the node may also be pruned. Figure 6 shows the binary tree associated with the four queens problem after two levels have been processed. The two levels of the tree beneath the root node correspond to the first two columns of the chessboard. The number in the information field of each node denotes a square (row), within the specified column, upon which a queen may be placed. Thus the left-most path of the tree corresponds to the placement of queens on the first square of column one and on the third square of column two. Notice that the tree of Figure 3 contains sixteen nodes at level two whereas the tree of Figure 6 contains only six nodes. The reason for this is that, in processing the second level, only those alternatives that do not produce a conflict condition are attached to nodes in the first level, whereas the tree of Figure 3 shows all possible alternatives, including those that produce a conflict condition. After all levels have been processed, the tree is either empty, in which case no solution exists, or it contains a path corresponding to each solution.

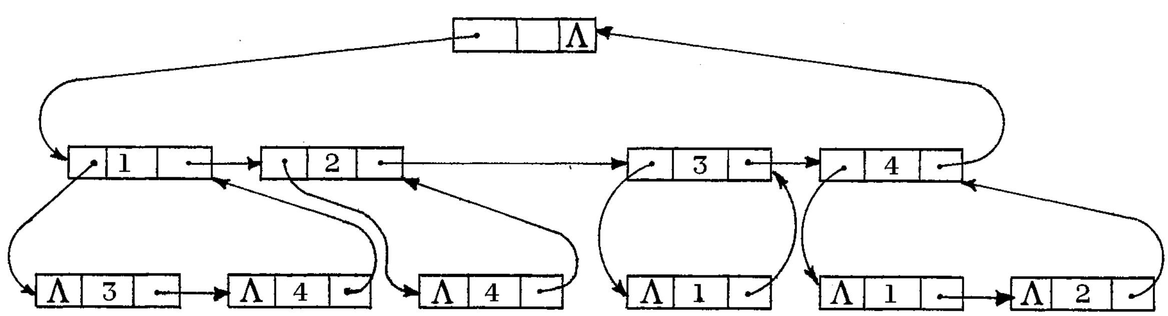


Figure 6. Binary Tree Associated with Four Queens Problem After Two Levels Have Been Processed. Since the root node is a dummy node, its information field is blank.

Two programs were written to find all solutions to the eight queens problem, one using the breadth-first approach, and the other using the depth-first approach. Both programs were written in Fortran IV for the IBM System 360 Model 65. Because of the combined effects of a low resolution timer and a multitasking environment, it was impossible to obtain accurate measurement of execution time; however, the execution times appear to be about the same for both methods, a surprising result when one considers the added overhead of storage management in the breadth-first approach. A major advantage of the depth-first approach is greater simplicity in programming, so it was decided to use this approach in investigation of the scheduling problem. A noteworthy advantage of the breadth-first approach is that, at the end of the procedure, all solutions are stored in a convenient structure, namely, the resultant binary tree. Also, the use of heuristic techniques of artificial intelligence in searching decision trees, which is suggested in Chapter VIII, may require a breadth-first traversal (7, 8).

For an excellent generalization of the concepts of decision trees and backtrack programming, see Golomb and Baumert (5).

CHAPTER III

SCHEDULING A SINGLE RESOURCE

The first scheduling problem to be investigated was that of scheduling a single resource unit. There are m activities that require the use of this resource. Associated with each of these activities is the actual length of time that the activity requires use of the resource, and one or more windows of time, that is, time intervals specified by a starting and ending time, during which the activity must be scheduled. The problem is to find a schedule for the resource such that every activity may use the resource during one of its windows for the length of time required, and that the resource is used by, at most, one activity at any instant of time. A sample problem with three activities is shown in Table I.

TABLE I

SAMPLE PROBLEM--SCHEDULING A SINGLE RESOURCE UNIT

Activity	Time Required	Windows
x_1	1 hour	1:00-3:00; 6:00-7:00
x_2	2 hours	2:00-4:00; 6:00-9:00
x_3	1 hour	8:00-9:00

It was observed in Chapter II that the eight queens problem could be reduced to the problem of selecting a square from each column such that no constraints are violated. By analogy, this scheduling problem can be reduced to selecting one window from the list of windows for each activity such that no constraints are violated. The determination of whether constraints are violated is somewhat more complex than in the eight queens problem. Suppose there are n intervals on the real line, corresponding to one window for each of n activities. These intervals are denoted by $[a_i, b_i]$ for i = 1 to n. Associated with each interval is some number, denoted by Δ_i, which corresponds to the actual time required by each activity. The problem of determining whether constraints are violated is equivalent to the problem of finding mutually disjoint subintervals $[c_i, d_i]$ such that for i = 1 to n

(1) $[c_i, d_i]$ is a subinterval of $[a_i, b_i]$, and

(2) $d_i - c_i = \Delta_i$.

The basic approach in determining whether constraints are violated is to generate permutations of the selected windows, and, for each permutation generated, attempt to schedule each activity as early in its window as possible, starting with the first window in the permutation. No activity can be scheduled prior to the start time of its window or prior to the completion of the previous activity. Let $[a'_i, b'_i]$ be the i'th window of the permutation currently being examined. Then,

(1) $c'_1 = a'_1$

(2) $c'_i = \max(a'_i, d'_{i-1})$ for i = 2 to n, and

(3) $d'_i = c'_i + \Delta'_i$ for i = 1 to n.

If d'_i exceeds b'_i for any i than the i'th activity cannot be scheduled within its window in the permutation currently being examined. If no permutation is found for which each activity can be scheduled within its window, then the choice of windows must be altered. A tree structured approach is used both in selecting windows and in generating permutations, as will be seen in the following paragraphs.

A program was written in PL/I for the IBM System 360 Model 65 which finds all combinations of windows (where one window is selected from the list of windows associated with each activity) for which a schedule exists. For each such combination, the program reports one possible schedule. In the same problem of Table I, there are four combinations of windows. Schedules exist for three of these combinations. A schedule for each of these three combinations is shown in Table II.

As stated previously, only one schedule per combination of windows is reported. Of course, there may be many schedules for each combination: (1) There may be more than one permutation of mutually disjoint subintervals; (2) if the time domain is considered to be a continuum, and if a subinterval, $[c'_i, d'_i]$, has the properties that $d'_i < b'_i$ and $d'_i < c'_{i+1}$, then an infinite number of schedules exist. Consider, for example, activity x_1 in the second schedule of Table II. This activity may be scheduled for 1:00-2:00, 1:01-2:01, 1:05-2:05, 1:15-2:15, and so forth. Even if a small finite resolution were imposed on the time domain, it would be combinatorially infeasible in most cases to examine and report all solutions. Therefore, the scope of the problem is limited to finding a sequence in which the activities can be scheduled, and finding a time interval in which each activity can be scheduled within that sequence.

TABLE II

THREE SCHEDULES FOR THE SAMPLE PROBLEM OF TABLE I

Activity	Window	Scheduled Time
Schedule 1		
x_1	1:00-3:00	1:00-2:00
x_2	2:00-4:00	2:00-4:00
x_3	8:00-9:00	8:00-9:00
Schedule 2		
x_1	1:00-3:00	1:00-2:00
x_2	6:00-9:00	6:00-8:00
x_3	8:00-9:00	8:00-9:00
Schedule 3		
x_2	2:00-4:00	2:00-4:00
x_1	6:00-7:00	6:00-7:00
x_3	8:00-9:00	8:00-9:00

The program contains an array of structures in which each structure corresponds to an activity. The information included in each structure includes the name of the activity, the actual time required, and the start and end time of each window associated with that activity. Figure 7 shows the array of structures corresponding to the sample problem of Table I. (The number of activities to be scheduled as well as the maximum number of windows per activity are input parameters which are used in allocating storage for this array.) This array is searched in a tree structured fashion using the depth-first approach

x_1	1	1	3	6	7
x_2	2	2	4	6	9
x_3	1	8	9	0	0

Figure 7. Internal Array of Structures Corresponding to the Sample Problem of Table I.

described in Chapter II. Each level of the tree corresponds to an activity and each node within a level corresponds to a window associated with that activity. As each node is visited, a pointer to the associated activity and window is placed on a pushdown stack, and a subprogram, CONFL1, is called to determine whether a schedule exists for the nodes (windows) on the stack. (Henceforth, the terms window and activity will be used interchangeably to denote items on the stack.) If a conflict condition is detected (that is, if no schedule can be found for the windows on the stack), then the search proceeds to the next window for the current activity, or, if all windows for the current activity have been examined, the search backtracks one level to the previous activity. If no conflict condition is detected, the search advances to the next level starting at the first window on that level, or, if all levels have been examined, reports that a solution has been found and advances to the next window of the current activity.

This tree structured search may be summarized as follows:

(1) Set level = 1.

(2) Set node = 1.

(3) Push node onto stack and call CONFL1.

(4) Has a conflict condition been detected? If so, go to step 7.

(5) Is this the last level? If so, a schedule has been found. Report the solution and go to step 7. Otherwise, continue.

(6) Add 1 to level. Go to step 2.

(7) Is this the last node at this level? If so, go to step 9.

(8) Pop node from stack. Add 1 to node. Go to step 3.

(9) If level = 1, then stop. Otherwise, subtract 1 from level, pop node from stack, and go to step 7.

CONFL1 is a subprogram whose calling parameter is the pushdown stack generated during the search of the window tree. This routine generates permutations of the items in the stack in lexicographical order, starting with the order in which the items appear in the stack. For each permutation generated, a call is made to another subprogram, CONFL2, which determines whether the activities can be scheduled in the order represented by the current permutation. If a permutation is found for which a schedule exists, then CONFL1 immediately returns control to the main program reporting a "no conflict" condition. If all permutations have been generated and no permutation has been found for which a schedule exists, then a conflict condition is returned to the main program.

Permutations are generated and examined in a manner corresponding to a depth-first, left to right tree search. For example, permutations of the numbers 1, 2, 3, and 4 may be represented by the tree shown in Figure 8. The leaves of this tree are, from left to right, all the permutations of the numbers 1, 2, 3, and 4 in lexicographical order. Permutations are generated one element at a time and calls are made to

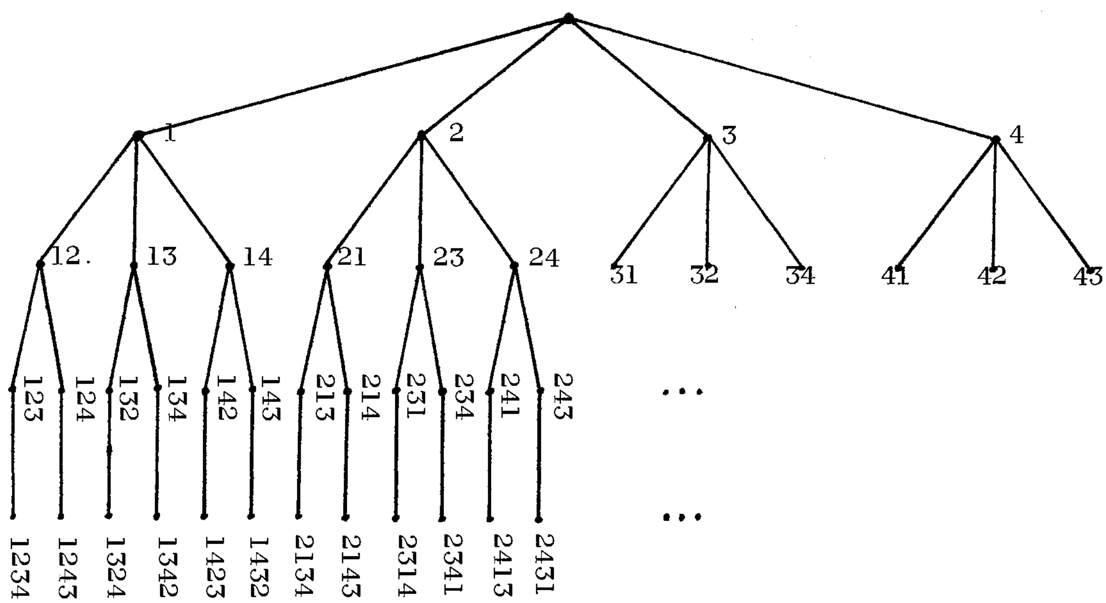


Figure 8. Permutation Tree.

CONFL2 to check the partial permutations being formed. If the activities represented in the partial permutation cannot be scheduled in the order specified by the permutation, then examination of the corresponding full permutations is precluded. For example, suppose four windows, denoted by w_1, w_2, w_3 and w_4 appear on the stack. The first call to CONFL2 is made with the partial permutation w_1, w_2. If it is found that the activities associated with w_1 and w_2 cannot be scheduled in the specified order, then it is not necessary to examine either of the permutations w_1, w_2, w_3, w_4, and w_1, w_2, w_4, and w_3. Although well-known algorithms exist for generating permutations in lexicographical order (9, 10, 11), no algorithms which would allow this preclusion capability were readily available. More will be said in Chapter VI regarding permutations.

CONFL2 is a subprogram whose calling argument is the current partial or complete permutation generated in CONFL1. This routine attempts to build a table of actual starting and ending times for the activities represented in the permutation, scheduling each activity as early in its window as possible. The starting and ending times in this table correspond to the mutually disjoint subintervals, denoted by $[c_i, d_i]$, referred to earlier in this chapter. An example may be found in Table III.

TABLE III

SAMPLE TABLE OF ACTUAL STARTING AND ENDING TIMES

Activity	Window (a_i, b_i)	Time Required (Δ_i)	Actual Time (c_i, d_i)
1	1-3	2	1-3
2	2-5	1	3-4
3	5-8	2	5-7
4	6-8	1	7-8

Although the program is not concerned with finding an optimal schedule, it may be enlightening at this point to consider possible criteria of optimality. Two possible goals would be to finish utilization of the resource at the earliest time possible or to begin utilization at the latest time possible. Other goals might be a "most dense" solution, in which the time from the start of the first activity to the end of the last activity is minimized, or a "most distributed" solution which is imprecisely defined but which will in some sense impose a uniform distribution of activity assignments over a period of time. Another way of describing a "most distributed" goal is that in which the total idle time for the resource is distributed evenly among the time intervals between activity assignments.

Once a goal has been chosen, one might ask whether it is possible to find an optimal schedule without examining all possible schedules. For example, suppose the goal is to find the "earliest schedule", that is, a schedule in which utilization of activities is completed as early as possible. One might suppose that by ordering the windows by increasing order of window start time, the first schedule found might be the earliest schedule, or might, at least, have some sort of "earliest" attribute. This question gives rise to the general question of ordering the windows in such a way that the optimal solution will be found as quickly as possible.

Another question that might be raised is whether the windows can be ordered in such a way that a schedule (not necessarily optimal) can be found as quickly as possible. Two possibilities for such an ordering are by increasing order of window start time or by decreasing order of time constraint, that is, by increasing order of $b_i - a_i - \Delta_i$. These questions will not be investigated any further in this report, but hopefully, they will provide the source for future investigations.

The remaining programs described in this report all have the same general structure as this one; that is to say, each program consists of a main program which traverses a decision tree of activities and windows, a subprogram named CONFL1 which generates permutations of activities, and a subprogram named CONFL2 which attempts to schedule the activities in the order specified by the permutation.

CHAPTER IV

SCHEDULING A SINGLE CLASS OF RESOURCES

The next problem investigated was that of scheduling a single resource class. A resource class consists of q 0 identical resource units. The resource units are identical in the sense that a request made for a unit of the specified class may be satisfied by any of the units within the class. Each activity to be scheduled requires exactly one unit of the resource class. The problem is to schedule each activity within one of its windows for its specified time required, in such a way that each resource unit is assigned to not more than one activity at any instant of time. Notice that two activities can be scheduled at the same time if there are two or more units in the class.

One could approach the problem with at least two different goals in mind. One of these goals is to minimize the number of resource units actually utilized. This goal would be employed in a problem where q units could be made available, but where it would be desirable to schedule all activities with fewer than q resource units. If all activities can be scheduled during mutually disjoint time intervals, then only one resource unit is required. Two activities are said to overlap if their actual scheduled times are not disjoint. For example, if activity one is scheduled for 4:00 to 7:00 and activity two is scheduled for 6:00 to 9:00, then activities one and two overlap. If all activities cannot be scheduled during mutually disjoint time intervals, then the number of units required does not exceed the maximum number of activities which overlap at any instant of time. In the schedule shown in Table IV three activities are scheduled during 6:00 to 7:00; therefore, three resource units must be available.

TABLE IV

A SCHEDULE REQUIRING THREE RESOURCE UNITS

Activity	Actual Time Scheduled
1	3:00-5:00
2	4:00-6:00
3	5:00-7:00
4	6:00-8:00
5	6:00-9:00

Another goal is to achieve a most uniformly distributed utilization among the resource units. This goal would be employed in a situation where q units would definitely be available and where it would be desirable to equalize utilization among the q units. It was decided to use this goal in the current investigation; its implementation will be described below.

There are two ways of viewing the search process in terms of decision trees. In one view, there are two levels in the tree per activity; one level contains nodes corresponding to the associated time windows, and the other level contains nodes corresponding to the resource units. Figure 9 shows such a tree for two activities, two windows per activity, and three resource units. This approach might be taken if it is desired to examine the effects of allocating

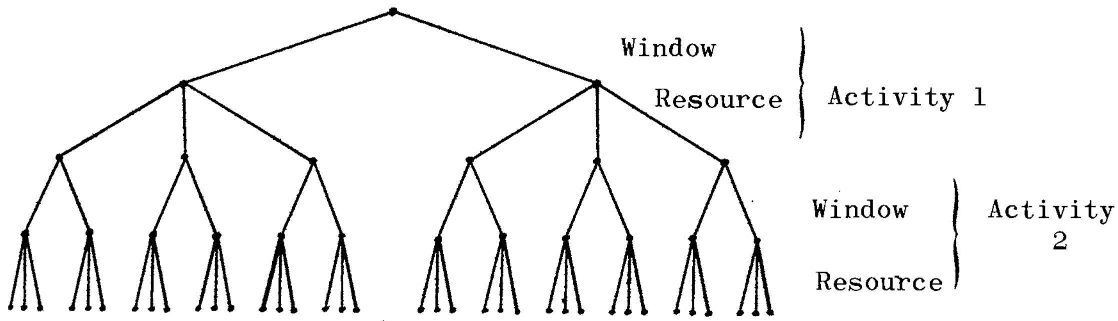


Figure 9. Decision Tree with Two Levels Per Activity.

different resource units to different activities. However, as can be seen by examining Figure 9, the combinatorial complexity of the problem proliferates greatly, even for a fairly small problem.

Another view is to have one level in the tree per activity, and in traversing the tree, allow the conflict checking routines to determine which resource unit, if any, can be allocated to an activity. This view can be taken if the resource class is viewed as a pool of identical resource units, and if it is immaterial, in terms of scheduling, which unit is allocated to a particular activity. It seems reasonable to expect that this approach would result in a shorter search time, especially if the method of unit selection were kept reasonbly simple.

The program described in Chapter III was modified, incorporating the second approach to the tree structured decision making process described above, so that it would handle a single class of resource units. The number of units available, q, is a required input parameter. The greatest number of changes were made in the CONFL2 subprogram. Firstly, the table of actual start and end times was expanded to include the number of the resource unit allocated. In addition, a pushdown stack is required for each resource unit, in which the top item indicates the start and end time of the latest allocation of that unit. Examination of the top item of a stack tells the earliest time that unit will be available for further allocation. Examples of the expanded table and corresponding stacks are shown in Table V.

TABLE V

SCHEDULE TABLE AND ASSOCIATED PUSHDOWN STACKS

Table of start and end times and unit allocated

Activity	Start	End	Unit
1	1:00	3:00	1
2	2:00	5:00	2
3	3:00	9:00	3
4	4:00	6:00	1
5	5:00	7:00	2

Associated Pushdown Stacks

Unit #1	Unit #2	Unit #3
1:00-3:00	2:00-5:00	3:00-9:00
4:00-6:00	5:00-7:00	

The reason pushdown stacks are required merits some further explanation. Recall that permutations are generated in a tree structured manner as described in Chapter III. In general, the use of a tree structured decision making process requires backtracking capability. Specifically, suppose there are eight activities to be scheduled, and Table V represents a schedule for the first five items in the schedule, that is to say, a choice has been made at level five in the permutation tree. Further suppose that each of the remaining three activities must begin before 6:00, which is the earliest time that a resource unit will be available. No branch can be taken from the current node at level five; therefore, the next alternative at level five, that is, the next partial permutation of five items in lexicographical order, must be examined. The start and end time in the fifth row of the table must be removed and the stack corresponding to resource unit two must be popped to indicate that unit two is no longer allocated for 5:00 to 7:00.

A circular polling mechanism is used in deciding which resource unit to assign to the next activity in the permutation. Suppose unit i was the last unit allocated to an activity, and it is desired to allocate a unit for the next activity in the permutation. The search for an available unit begins with unit i + 1, proceeds to unit q, then proceeds from unit 1 to unit i. This is roughly equivalent to maintaining a first-in, first-out queue of resource units, where a unit is returned to the end of the queue when an activity has finished using it. This circular polling method is used because in most cases a more distributed allocation can be expected from this method than from a method which always begins searching at unit 1.

Perhaps the program described here could be modified so that it could determine the minimum number of resource units required. This is a question that will be left for future investigation.

CHAPTER V

SCHEDULING MULTIPLE RESOURCE CLASSES

In this chapter we consider the problem of scheduling m activities on n different resource classes. Each resource class, y_i, contains q_i units. Each activity may require exactly one unit of one or more resource classes. Specifications for each activity include actual time required, windows of time, and a list of resource classes of which a unit is required. It is assumed that all resources required by an activity are to be assigned during the same time interval. Specifications for each resource class include the number of resource units in the class. A sample problem is shown in Table VI.

Extending the scope of the problem from one resource class to n resource classes increases the combinatorial complexity of the problem in terms of the number of alternatives to be examined. One way to reduce this complexity is to identify subsets of activities in such a way that each subset may be scheduled independently of the other subsets. If there are 10 activities to be scheduled with two windows per activity, the number of leaves in the decision tree corresponding to the activities and their windows (which will henceforth be referred to as the window tree) is 2^{10} or 1024. However, if two subsets of five activities each could be identified, the search could be reduced to two window trees each of which contains 2^5 or 32 leaves.

TABLE VI

SAMPLE PROBLEM FOR MULTIPLE RESOURCE SCHEDULING

Resource Class	Number of Units
y_1	2
y_2	1
y_3	5
y_4	8
y_5	6
y_6	3

Activity	Time Required	Windows	Resource Classes
x_1	2	7-9; 10-12	y_1, y_2
x_2	1	1-2; 5-6	y_2, y_3
x_3	1	3-4	y_4
x_4	2	2-5	y_6
x_5	3	1-7	y_3, y_5
x_6	1	1-3; 9-12	y_4, y_6

Consider an undirected graph in which each node corresponds to an activity and in which an arc from node i to node j indicates that activities x_i and x_j share a common requirement for at least one resource class. A graph for the sample problem of Table VI is shown in Figure 10. Each connected component of such a graph identifies a subset of activities which must be scheduled interdependently. In this sample problem activities x_1, x_2, and x_5 collectively require units from resource classes y_1, y_2, y_3, and y_5, and activities x_3, x_4,

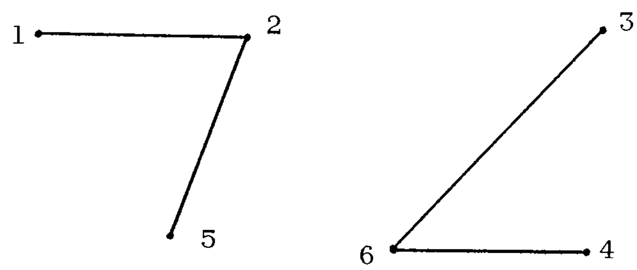


Figure 10. Graph Showing Common Resource Requirements Among Activities.

and x_6 collectively require units from resource classes y_4 and y_6. Evidently, activities x_1, x_2 and x_5 can be scheduled independently of activities x_3, x_4 and x_6 because allocation of resource units to x_1, x_2 and x_5 would have no effect on the availability of resource units for x_3, x_4, and x_6.

The connected components of the graph described above are identified as follows. The adjacency matrix is constructed, then an algorithm by Warshall (12) is employed to construct the path matrix. A distinct row value of the path matrix defines a connected component of the graph, and therefore, a subset of activities. Figure 11 shows the adjacency and path matrices for the graph in Figure 10. There are two distinct row values in the path matrix.

The program described in Chapter IV actually assigns individual resource units to activities. In contrast, the approach taken here is to determine the number of units of each resource class that are required at any instant of time and to determine whether each resource class has enough units to meet those requirements. In order to reduce the combinatorial complexity of the problem, it was decided not to make assignments of individual units.

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	0	0	0	1	0
3	0	0	0	0	0	1
4	0	0	0	0	0	1
5	0	1	0	0	0	0
6	0	0	1	1	0	0

Adjacency Matrix

	1	2	3	4	5	6
1	1	1	0	0	1	0
2	1	1	0	0	1	0
3	0	0	1	1	0	1
4	0	0	1	1	0	1
5	1	1	0	0	1	0
6	0	0	1	1	0	1

Path Matrix

Figure 11. Adjacency and Path Matrices for Graph in Figure 10.

In this program, the CONFL2 subprogram still attempts to schedule each activity as early in its window as possible. There is a table of start and end times for the activities scheduled; also for each resource class there is a corresponding table of start and end times for the activities requiring units of that resource class. In scheduling activities x_1, x_2, and x_5, these tables might appear as in Table VII. When attempting to schedule the next activity in the permutation tree, the table corresponding to each resource class required by the next activity is examined to determine the earliest time (greater than or equal to the window start time) that a unit of that resource will become available. This is done by counting the number of activities whose scheduled times overlap the proposed scheduled time of the current activity, and comparing that count against the number of units in the resource class. A previously scheduled activity is presumed to overlap the activity currently being scheduled if the ending time of the previously scheduled activity exceeds the window start time of the current activity. This is a rather restrictive presumption which may result in no schedule being found when a schedule actually exists. A better method of counting overlapping activities will be presented in Chapter VII.

TABLE VII

TABLES OF START AND END TIMES FOR EACH RESOURCE CLASS

All	y_1	y_2	y_3	y_5
7-9	7-9	7-9	1-2	2-5
1-2		1-2	2-5	
2-5				

After the earliest available time for each resource class has been determined, the latest of these times is taken to be the actual starting time of the activity being scheduled. The actual time required is added to the starting time to give the actual ending time. If the actual ending time exceeds the window end time, then the activity cannot be scheduled within its window.

A schedule produced by this program shows, for each resource class, the exact times that resource units are to be assigned to activities. Furthermore, the approach taken guarantees that the assignments can be made. Once a schedule has been produced, a circular polling mechanism, similar to the one described in Chapter IV, could be employed to make assignments of individual units.

CHAPTER VI

EFFICIENCY IN GENERATING AND EXAMINING PERMUTATIONS

During the course of testing the program described in Chapter V, it became evident that increased speed in generating and examining permutations of activities was necessary. The present chapter is concerned with possible improvements in that direction, and describes the improvements that were actually implemented.

Whenever a new node in the window tree is visited, a pointer to that activity and window is placed on a stack, and a call is made to CONFL1 in an attempt to find a schedule for all activities which have pointers on the stack. CONFL1 generates permutations of the pointers on the stack and, for each permutation generated, calls CONFL2, which attempts to schedule the activities in the order specified by the permutation. These permutations are generated in a depth-first tree searching manner; one may speak of traversing a tree of permutations.

The permutations are generated in lexicographical order. Knuth (6) shows two other methods of generating permutations; however, one advantage of lexicographical ordering is that information gained in scheduling the previous permutation can be used in scheduling the current permutation. If the current permutation consists of n elements, then it can be assumed that a schedule has already been found for the first n - 1 elements in the permutation. For example, consider a call to CONFL2 made with a partial permutation 31425. Due to the nature of depth-first tree traversal, it can be assumed that the activities corresponding to the partial permutation 3142 have already been scheduled; furthermore, the schedule for 3142 is retained in CONFL2, so all that is necessary is to schedule the activity corresponding to 5.

When a new node in the window tree is examined, the entire process of generating permutations is repeated from the beginning. The question to be examined is how can information gained from the previous call to CONFL1 be retained, and how can this information be used to hasten the current permutation check. It would be desirable to eliminate some permutations from consideration based on the fact that similar permutations failed to produce a schedule in a previous call to CONFL2.

Consider one possible example. Suppose four activities are represented in the stack and a fifth activity is being added. Of the four activities, originally in the stack, suppose that the first permutation, in lexicographical order, that produced a schedule was 3142. Considering permutations of five activities, it is evident that 12345 will not produce a schedule, because if 12345 were to produce a schedule, then 1234 would have produced a schedule for four activities. Indeed, the first permutation that need be considered is 31425. Also, permutations such as 31524, 51234, 52431 can be removed from consideration for reasons explained below.

As another possibility, suppose there are two activities represented in the stack, and the permutation 1,2 does not produce a schedule but the permutation 2,1 does. It is evident that 2 must precede 1 in any permutation that contains both 1 and 2. It might be desirable to find all pairs of activities in which one activity must precede the other before beginning to generate permutations. Perhaps this idea could be generalized, and necessary ordering relationships among triplets, quadruplets, and so forth, could be found. This would correspond to a breadth-first search of the first few levels of the permutation tree, coupled with a depth-first search of the remainder of the tree.

Two changes were made to the program described in Chapter V with respect to generating and checking permutations. Firstly, corresponding to each level in the window tree, a record is kept of the permutation that produced a schedule at that level. When a node at level i in the window tree is visited, permutations are generated beginning with the permutation stored for level i - 1. Secondly, each new permutation generated at level i in the window tree is compared to the permutation stored for level i - 1 to detect violations of lexical ordering. For example, suppose 3142 is the permutation stored for level four, and while processing level five in the window tree, the permutation 31524 is generated. Since 3124 precedes 3142 in lexicographical ordering, the permutation 3124 cannot produce a schedule because if 3124 could produce a schedule, then 3124 would have been stored for level four. Since 3124 cannot produce a schedule, then 31524 cannot produce a schedule either. This can be proved as follows. Suppose a schedule is found for 31524, which would mean that the activities could be scheduled in the order specified by the permutation 31524. If one of these activities, say activity 5, is eliminated, the remaining four activities could still be scheduled in the specified order. However, it is known that the permutation 3124 did not produce a schedule. Therefore, it can be concluded that 31524 cannot produce a schedule; hence 31524 can be eliminated from consideration.

Further possibilities for improvement, such as recognition of problem decomposition at various levels in the permutation tree, are pointed out by Bratley, et. al. (3).

CHAPTER VII

SELECTING RESOURCES BASED ON ATTRIBUTES

The program described in this chapter extends the flexibility of resource class selection and requirement specification by allowing attributes to be specified for each resource class, thus associating each resource class with one or more attribute groups, and allowing resource requirements to be specified in terms of attribute groups rather than specific resource classes. When an activity requires a resource unit of a specific attribute group, that unit may be selected from any resource class which is a member of the specified attribute group. A resource unit may service at most one requirement at any one time, but it may service requirements for different attribute groups at different times. The ability to service requirements for different attribute groups at different times has been restricted in the present implementation for reasons explained below.

As an example, suppose there are seven resource classes, denoted by y_j for j = 1 to 7, and three attribute groups, denoted by A_1, A_2, and A_3. In an airline scheduling problem, for example, there might be seven different kinds of aircraft used by the airline. Attribute group A_1 might consist of all aircraft with seating capacity greater than 120, attribute group A_2 might consist of all jet powered aircraft, and attribute group A_3 might consist of all aircraft that can land on a 5,000 foot runway. Figure 12 shows a possible association between resource classes and attribute groups. A request for a unit of group A_2, for example, could be satisfied by a unit of one of the resource classes y_2, y_3, y_5, y_7. Units in class y_1 may satisfy requests for group A_1 whereas units of class y_4 may satisfy requests for either A_1 or A_3.

Attribute Groups	Resource Classes						
	y_1	y_2	y_3	y_4	y_5	y_6	y_7
A_1	X		X	X			
A_2		X	X		X		X
A_3			X	X		X	

Figure 12. Association Between Resource Classes and Attribute Groups.

Subsets of activities that can be scheduled independently can be determined by the same graph theoretic method as was used in the program described in Chapter VI. In this case an arc is drawn between two nodes if the activities corresponding to the two nodes share at least one common attribute group requirement.

Let q_j be the number of units of class y_j and $R(A_i)$ be the number of units of group A_i required at some instant of time. It is desired to determine whether there exists an assignment of resource units which satisfies the following conditions:

(1) The number of units assigned to satisfy the requirements of each group, A_i, is $R(A_i)$.

(2) The number of units assigned from each class y_j does not exceed q_j.

(3) A resource unit which is a member of class y_j is assigned to group A_i only if y_j is a member of A_i. (A unit is

assigned to group A_i if that unit is assigned to an activity which requires a unit of group A_i.)

This problem is a special case of the transportation problem of linear programming (13). In the transportation problem there are a specified number of suppliers, each of which can supply a specified number of units, and a specified number of customers, each of which must receive a specified number of units. Also there is a known cost of shipping a single unit from supplier i to customer j. The problem is to minimize the total shipping cost subject to the constraint that all customer demands be met.

To apply the transportation model to the resource assignment problem, one would consider the resource classes as suppliers and the attribute groups as customers. The cost of assigning a unit of resource class y_j to satisfy a requirement for A_i is zero if resource class y_j is a member of attribute group A_i and is one otherwise. The analogy between the general transportation problem and the resource assignment problem is shown in Table VIII. Bayer's transportation algorithm (14) is used to find an assignment that minimizes the total cost. The assignment can be made only if the minimized total cost is zero.

The next problem to be considered is the determination of requirements for each attribute group during a given time interval, and the use of the transportation algorithm in the CONFL2 subprogram to determine whether the next activity can be scheduled. Suppose a call is made to CONFL2 with n activities in the permutation. As explained in Chapter VI, the first n - 1 activities have been scheduled so that the task at hand is to schedule the n'th activity. The scheduled start and

TABLE VIII

ANALOGY BETWEEN GENERAL TRANSPORTATION PROBLEM AND RESOURCE ASSIGNMENT PROBLEM

Transportation Problem	Resource Assignment Problem
Suppliers	Resource Classes
Customers	Attribute Groups
Shipping Cost	"Cost" is 0 or 1

end times for the first n - 1 activities have been retained in the tables described in Chapter V. Let t_0 and t_1 be the proposed start and end times for activity n. Initially let t_0 be equal to the window start time for activity n. Then proceed as follows:

(1) Compute t_1 by adding the actual time required by activity n to t_0.

(2) Compute the number of units of each attribute group required by the n activities during the time interval bounded by t_0 and t_1. A procedure used for this computation is described below.

(3) Invoke the transportation algorithm. If the minimized total cost is zero, then the attribute requirements can be satisfied during the time interval bounded by t_0 and t_1, and t_0 and t_1 are entered as the scheduled start and end times for activity n.

(4) If the minimized cost is greater than zero, then set t_0 equal to the earliest time that any attribute requirement may decrease. The earliest time any attribute requirement may

decrease is the earliest scheduled ending time of the first $n - 1$ activities. Recompute t_1, and if t_1 does not exceed the window end time, then return to step 2. Otherwise, report that activity n cannot be scheduled.

In the program described in Chapter V, a table was kept for each resource class, which contained scheduled start and end times of activities requiring units of that resource class. In this program, such a table is kept for each attribute group. It was noted in Chapter V that the method used for counting the number of overlapping activities was unduly restrictive. Suppose for example, the scheduled time for activity x_1 was 4:00 to 6:00, and the scheduled time for activity x_2 was 6:00 to 8:00. If the proposed scheduled time for activity x_3 was 5:00 to 7:00, the method used in the previous program would count two overlapping activities and conclude that three units were required, when it is clear that only two units are required. A more accurate method of determining the number of units of an attribute group required during a specified time interval is used in this program. For any attribute group, let k be the number of units required during the time interval bounded by t_0 and t_1, and let (c_1, d_1), (c_2, d_2), ..., (c_{n-1}, d_{n-1}) be the start and end times of those activities already scheduled which require a unit of that attribute group. Let f_1, f_2, ..., f_{n-1} be flags associated with each scheduled activity. Each flag will indicate whether the scheduled time of its corresponding activity overlaps the time interval bounded by t_0 and t_1. The value of k is computed as follows:

(1) Set k equal to zero. Set f_i equal to zero for all i.

(2) Order the c_i, d_i pairs in increasing order of c_i. Choose a value for j such that $c_{j-1} \leq t_0 \leq c_j$.

(3) This step counts the number of overlapping activities that begin before t_0. For k = 1 to j - 1, if $d_1 > t_0$, then set $f_i = 1$ and add 1 to k.

(4) This step counts the number of overlapping activities that begin after t_0. If two activities both overlap the interval being examined but do not overlap each other, then they may be counted as one activity. For i = j to n - 1:
If $c_i < t_1$ then for $\ell = 1$ to i - 1 search for a pair c_ℓ, d_ℓ where $f_\ell = 1$ and $d_\ell \leq c_i$. If such a pair is found, set $d_\ell = d_i$. Otherwise set $f_i = 1$ and add 1 to k.

An example is shown in Table IX. Note that the second and third activity both overlap the time period 3:00 to 5:00, but since they do not overlap each other, they may be considered as one activity scheduled for 2:00 to 6:00.

This method examines whether resource assignments can be made during sub-intervals of time, without considering whether or not assignments can be made for the entire period of time under consideration. Diabolical cases may arise in which the assignment can be made during each sub-interval but not for the entire period of time under consideration. An example of such a case is shown in Table X.

When the permutation consists of x_1, x_2, and x_3, the time interval under consideration is 9:00 to 11:00. The only assignment that could be made is two units of y_1 for A_1 and one unit of y_2 for A_2. When the permutation consists of x_1, x_2, x_3, x_4, and x_5, the time interval to be considered is 10:00 to 12:00. The only assignment that

TABLE IX

COMPUTATION OF THE NUMBER OF UNITS REQUIRED OF A PARTICULAR ATTRIBUTE GROUP

$t_0 = 3{:}00$	$t_1 = 5{:}00$	
$c_1 = 1{:}00$	$d_1 = 3{:}00$	$f_1 = 0$
$c_2 = 2{:}00$	$d_2 = 4{:}00$	$f_2 = 1$
$c_3 = 4{:}00$	$d_3 = 6{:}00$	$f_3 = 0$
$c_4 = 5{:}00$	$d_4 = 7{:}00$	$f_4 = 0$
$k = 1$		

TABLE X

A CASE FOR WHICH AN ASSIGNMENT CAN BE MADE FOR EACH SUBINTERVAL, BUT CANNOT BE MADE FOR THE ENTIRE PERIOD OF TIME

Activity	Window	Time Required	Attribute Groups Required
x_1	8:00-10:00	2	A_1
x_2	8:00-10:00	2	A_1
x_3	9:00-11:00	2	A_2
x_4	10:00-12:00	2	A_3
x_5	10:00-12:00	2	A_3

Resource Class	Attribute	Quantity
y_1	1, 2	2
y_2	2, 3	2

could be made is one unit of y_1 for A_2 and two units of y_2 for A_3. Notice that assignments can be made for each subinterval of time but that one unit cannot be assigned to x_3 continuously from 9:00 to 11:00. The method described above would report that a schedule exists when in fact no schedule can be found.

To avoid such situations, we add the restriction that a resource unit may be assigned to only one attribute group during the entire period of time under consideration. In the example of Table X, if a unit of y_1 were assigned to an activity requiring a unit of A_1 from 8:00 to 10:00, then the same unit could be assigned to another activity requiring a unit of A_1 after 10:00, but the unit could not be assigned to satisfy an activity's request for A_2 even though class y_1 is a member of group A_2. To implement this restriction, a dummy activity is added which requires no resources but which must be scheduled for the entire period of time under consideration. This forces CONFL2 to look for an assignment that can be made for the entire time period. In the example of Table X, an attempt to schedule a dummy activity during the time interval 8:00 to 12:00 would cause CONFL2 to report that no schedule could be found.

There are cases, however, for which this added restriction would cause a schedule not to be found when in fact a schedule exists. Suppose two activities request units of attribute group A_1; one of the activities can be scheduled from 8:00 to 10:00 and the other from 10:00 to 12:00. Suppose two resource classes, y_1 and y_2 can service the request, and that a unit of y_1 is available from 8:00 to 10:00 and a unit of y_2 is available from 10:00 to 12:00. Clearly a schedule exists, but the additional restriction described above may result in a report that no schedule can be found.

It was decided to take the more restrictive approach and use the dummy activity in the program at the cost of possibly not finding a schedule when one does exist. The problem of guaranteeing that a schedule will be found if and only if one does exist apparently remains unsolved at the time of this writing.

CHAPTER VIII

CONCLUSION AND SUGGESTIONS FOR FURTHER INVESTIGATION

The primary goal of this investigation has been the application of tree structured processes to the solution of a certain class of scheduling problems. This goal has been attained through the development of four computer programs. Three of these four programs were written to solve subclasses of the class of scheduling problems under consideration, and the fourth program was written to solve the full class of problems. Except for certain cases which are noted elsewhere in this report, each of these four programs solves the class or subclass of problems for which it was written. Another goal which has been achieved was the elimination of the need to impose a discrete resolution on the time dimension. This has been done by scheduling each activity as early in its window as possible.

In addition to the attainment of these goals, the investigation resulted in several other significant achievements. One of these is the use of graph theoretic techniques to identify independent subsets of activities, as described in Chapter V. Another accomplishment is the development of an algorithm to count the number of units of an attribute group required during a subinterval of time. Still another accomplishment is the application of a solution method for the transportation problem to the problem of assigning resource classes to attribute groups, as described in Chapter VII.

However, the author believes that the most important results of the investigation are to be found not in the goals that have been achieved, but in the problem areas that have been uncovered by the investigation which could lead to further study. Traversal of decision trees has been of primary importance in developing these programs. It may well be said that the investigation itself has proceeded in a tree structured manner. In a number of instances during the development of the above-mentioned programs, interesting problems and questions suitable for further investigation were encountered; in each case a decision had to be made as to whether to turn the investigation toward a deeper study of the problem uncovered or to continue in the current direction. In the following paragraphs, some unbeaten paths in this decision tree are outlined.

It was conjectured in Chapter III that, by ordering the windows in increasing order of window start time, the first schedule found would have some earliest attribute associated with it. The effect of ordering windows merits further investigation. Will ordering of windows in decreasing order of time constraint produce a solution in the shortest time by creating conflicts early in the decision making process? In each program the CONFL2 routine attempts to schedule each activity as early in its window as possible. If the windows were ordered by decreasing order of start time (or perhaps end time) and the CONFL2 routine were changed so that each activity was scheduled as late in its window as possible, would the first solution found be the "latest" solution?

The method of assigning resource units to attribute groups described in Chapter VII could use some improvement. An algorithm is used which can find a solution to the transportation problem in its full generality. It seems that a faster algorithm could be developed for this special case. Perhaps an algorithm could be developed which would determine whether the assignment could be made, and, if the assignment could not be made, would determine the minimum change in attribute requirements necessary for an assignment to be made.

Improvements with respect to generating and checking permutations were discussed in Chapter VI. For a large problem, it is evident that an enumeration of all permutations is combinatorially infeasible. Heuristic techniques need to be developed which will choose the "best" path in a decision tree, that is, the path that is most likely, in some respect, to arrive at a solution. The interested investigator is referred to Slagel and Lee (15) for a discussion of heuristic techniques applied to tree searching problems.

Lastly, the feasibility of applying the final program to a fairly large problem should be studied. Since this investigation has been concerned mainly with techniques and methods, no attempt has been made to determine the amount of time required to solve scheduling problems of various sizes. The problem shown in the sample output of Appendix B has nine activities, five resource classes, and eight attribute groups; no attempt has been made to test a larger problem. Variables that should be considered in such a study include the number of activities, the number of windows per activity, the severity of time and resource constraints, and the number of subsets of independent activities.

Hopefully, the techniques developed in this investigation, together with the results of further investigations, will be useful in the development of a non-procedural scheduling language which is expected to be undertaken locally in the near future.

BIBLIOGRAPHY

(1) Van Doren, J. R. "Space Flight Scheduling." Attachment 1 to Arts and Sciences Research Report on Project 3722-16/D-2160, Oklahoma State University, January, 1973.

(2) Muth, John F., and Gerald L. Thompson. Industrial Scheduling. Englewood Cliffs: Prentice-Hall, 1963.

(3) Bratley, Paul, Michael Florian, and Pierre Robillard. "Scheduling with Earliest Start and Due Date Constraints." Publication No. 56, University of Montreal, March, 1971.

(4) Davis, Edward W., and George E. Heidorn. "An Algorithm for Optimal Project Scheduling under Multiple Resource Constraints," Management Science, Vol. 17, 12 (1971), 803-816.

(5) Golomb, Solomon W., and Leonard D. Baumert. "Backtrack Programming," Journal of the ACM, Vol. 12 (1965), 516-524.

(6) Knuth, Donald E. The Art of Computer Programming, Vol. I. Reading: Addison-Wesley, 1968.

(7) Nilsson, Nils J. Problem Solving Methods in Artificial Intelligence. New York: McGraw-Hill, 1971.

(8) Slagel, James R. Artificial Intelligence: The Heuristic Programming Approach. New York: McGraw-Hill, 1971.

(9) Schrack, G. F., and M. Shimrat. "Algorithm 102, Permutations in Lexicographical Order," Communications of the ACM, Vol. 5 (June, 1962), 346.

(10) Shen, Mok-Kong. "Algorithm 202, Generation of Permutations in Lexicographical Order," Communications of the ACM, Vol. 6 (September, 1963), 517.

(11) Ord-Smith, R. J. "Algorithm 323, Generation of Permutations in Lexicographic Order," Communications of the ACM, Vol. 11 (February, 1968), 117.

(12) Warshall, Stephen. "A Theorem on Boolean Matrices," Journal of the ACM, Vol. 9 (1962), 11-12.

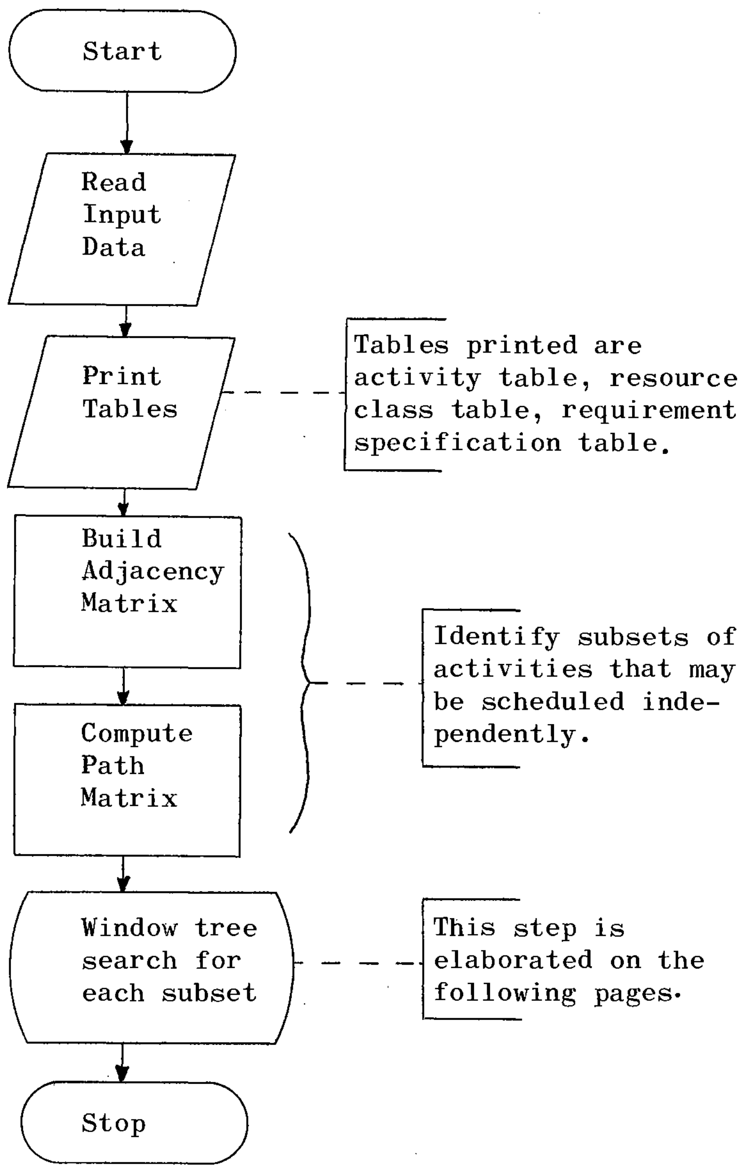
(13) Wagner, Harvey M. Principles of Operations Research. Englewood Cliffs: Prentice Hall, 1969.

(14) Bayer, G. "Algorithm 293, Transportation Problem," Communications of the ACM, Vol. 9 (December, 1966), 869-871.

(15) Slagel, James R., and Richard C. T. Lee. "Application of Game Tree Searching Techniques to Sequential Pattern Recognition," Communications of the ACM, Vol. 14 (February, 1971), 103-110.

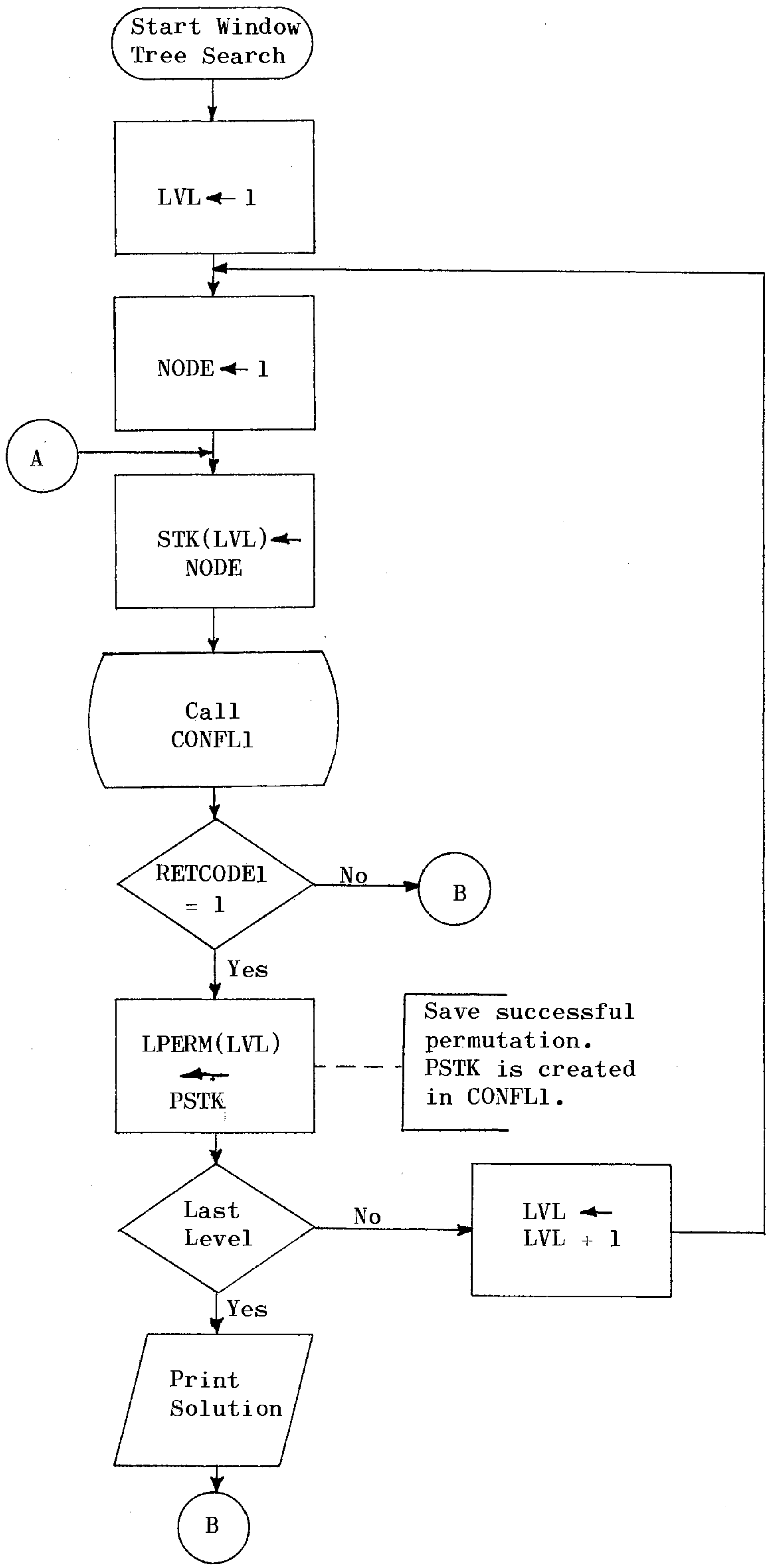
APPENDIX A

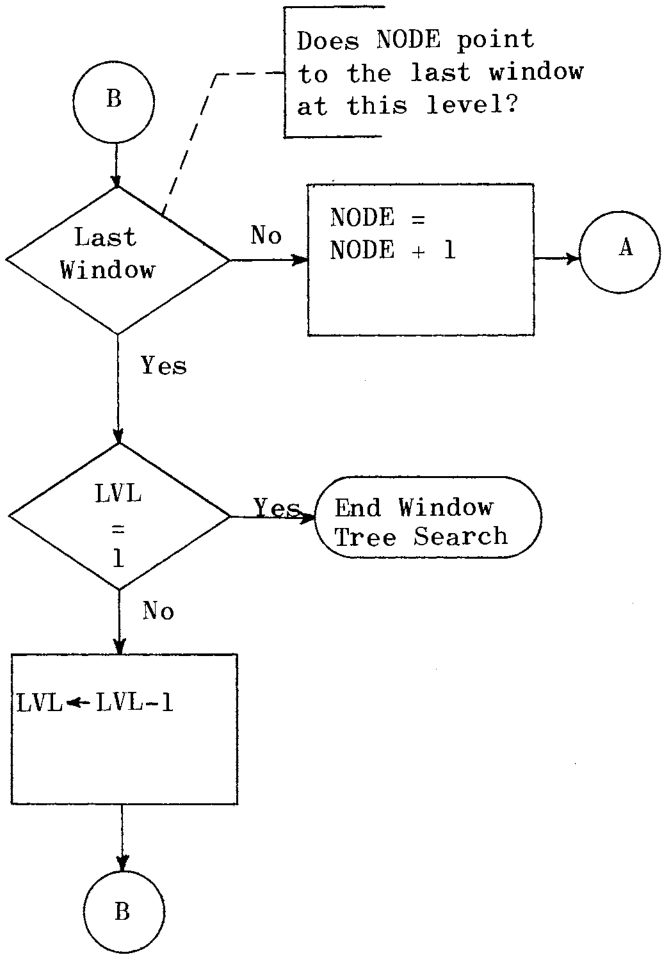
FLOWCHART OF FINAL PROGRAM

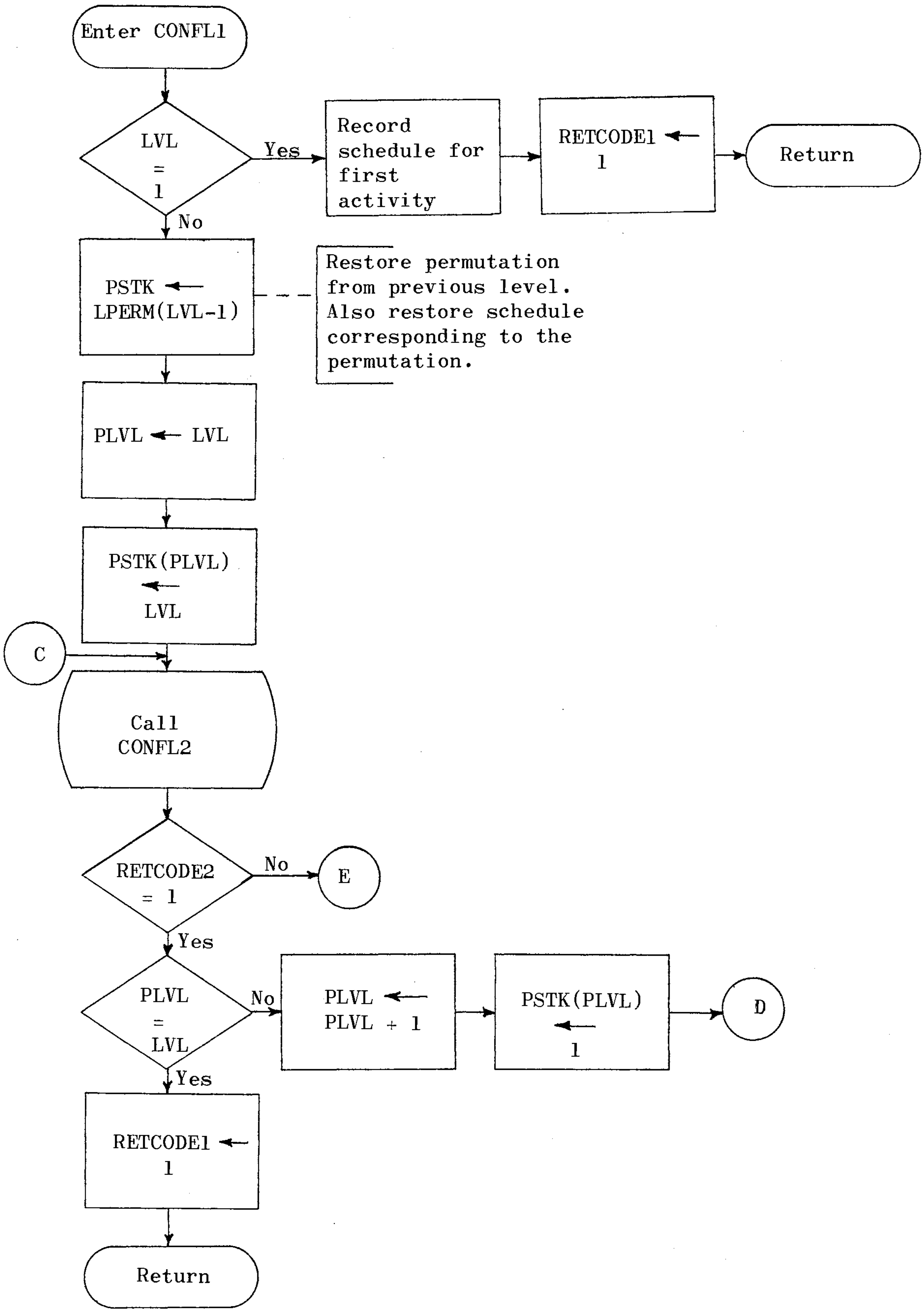


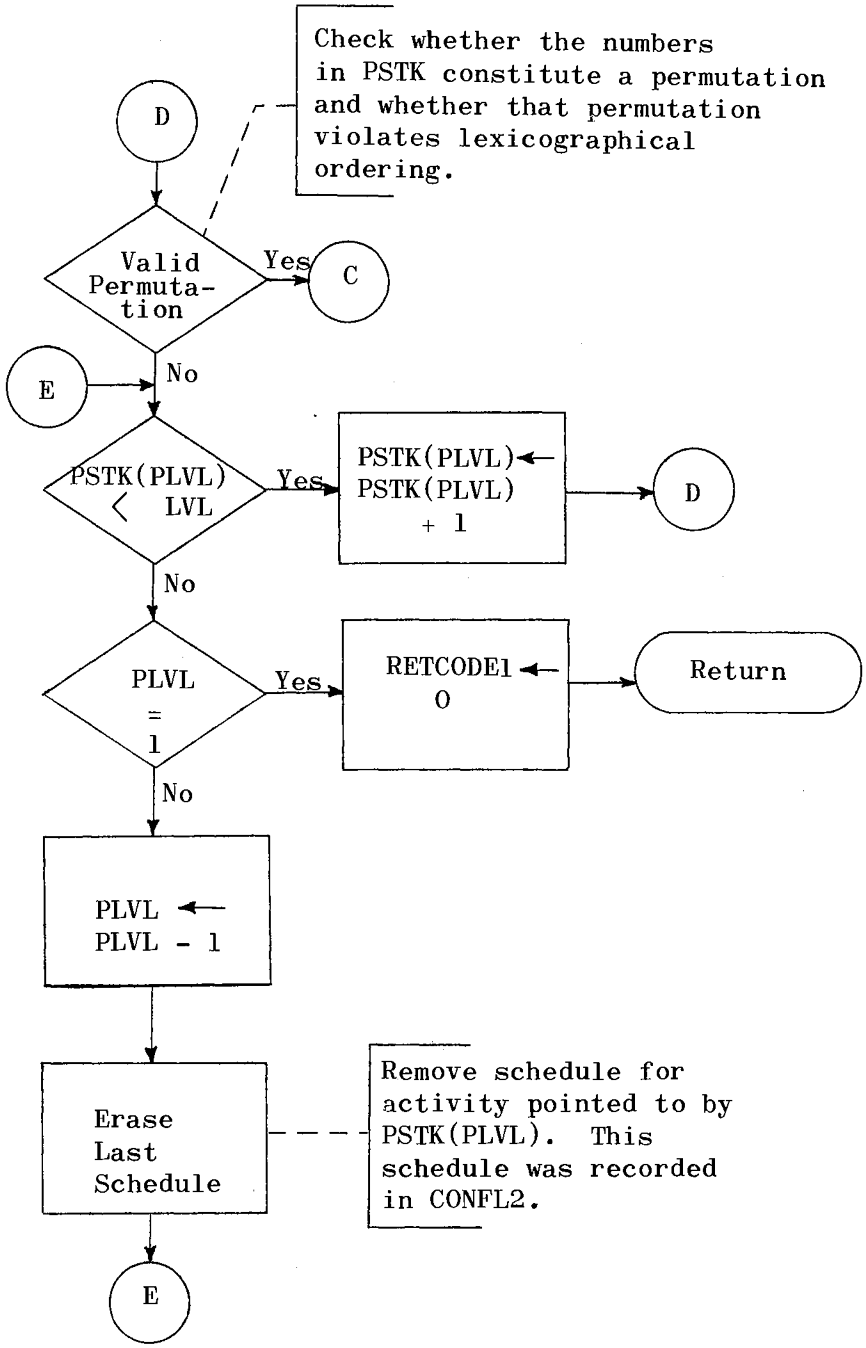
<u>Legend</u>

LPERM - save area for permutations
LVL - current level of window tree
NODE - pointer to window at current level
PLVL - current level of permutation tree
PSTK - vector containing permutation
RETCODE1 - return code set by CONFL1
RETCODE2 - return code set by CONFL2
SLS - tentative activity start time
SLE - tentative activity end time
STK - stack used in window tree traversal

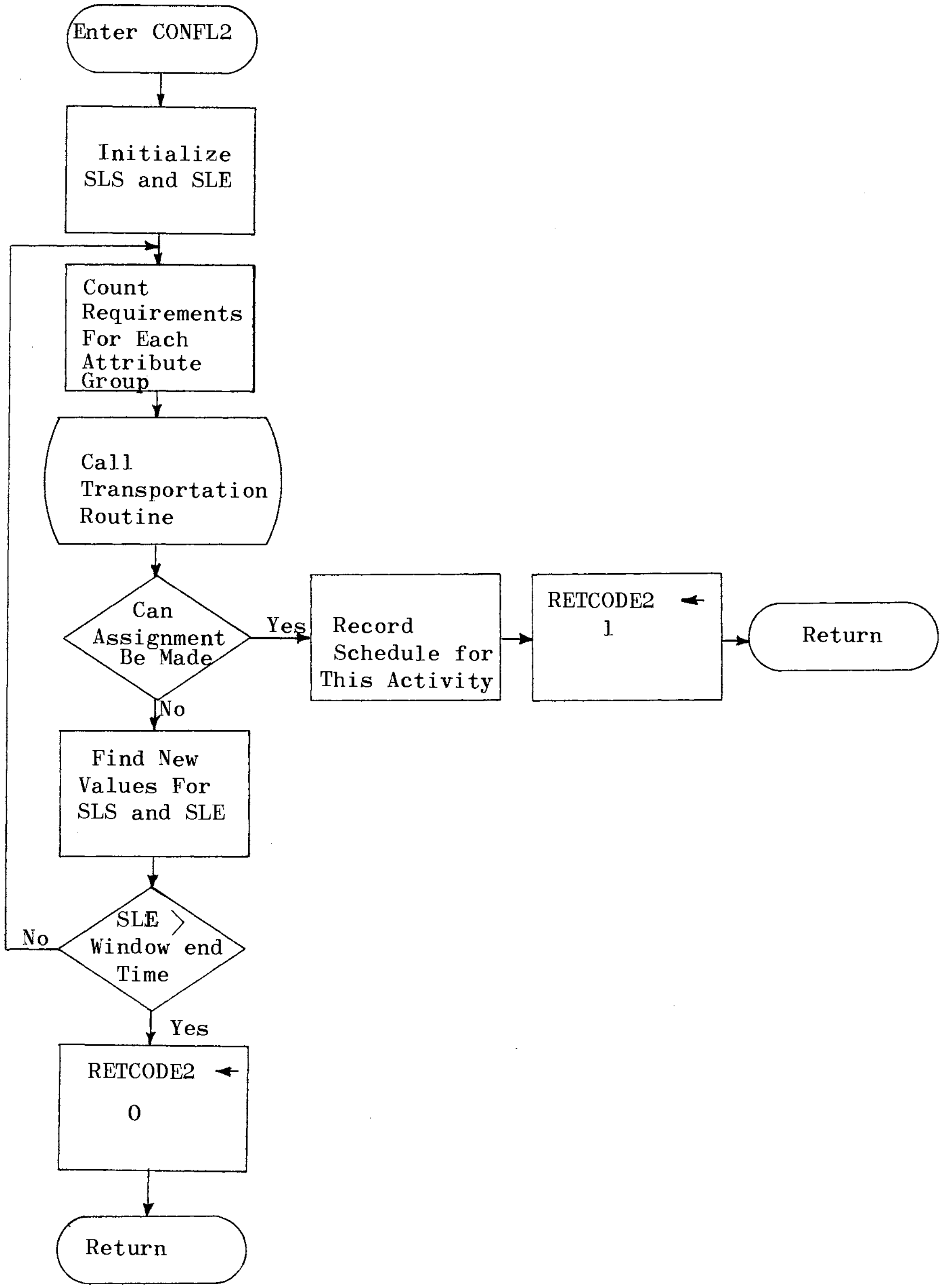

Start Window Tree Search
LVL ← 1
NODE ← 1
A
STK(LVL) ← NODE
Call CONFL1
RETCODE1 = 1
No
B
Yes
LPERM(LVL) ← PSTK
Save successful permutation. PSTK is created in CONFL1.
Last Level
No
LVL ← LVL + 1
Yes
Print Solution
B


Does NODE point
to the last window
at this level?
B
Last
Window
No
NODE =
NODE + 1
A
Yes
LVL
=
1
Yes
End Window
Tree Search
No
LVL ← LVL-1
B


Enter CONFL1
LVL = 1
Yes
No
Record schedule for first activity
RETCODE1 ← 1
Return
PSTK ← LPERM(LVL-1)
Restore permutation from previous level. Also restore schedule corresponding to the permutation.
PLVL ← LVL
PSTK(PLVL) ← LVL
C
Call CONFL2
RETCODE2 = 1
No
E
Yes
PLVL = LVL
No
PLVL ← PLVL + 1
PSTK(PLVL) ← 1
D
Yes
RETCODE1 ← 1
Return



Check whether the numbers in PSTK constitute a permutation and whether that permutation violates lexicographical ordering.
D
Valid Permuta-tion
Yes
C
No
E
PSTK(PLVL) < LVL
Yes
PSTK(PLVL) ← PSTK(PLVL) + 1
D
No
PLVL = 1
Yes
RETCODE1 ← 0
Return
No
PLVL ← PLVL - 1
Erase Last Schedule
Remove schedule for activity pointed to by PSTK(PLVL). This schedule was recorded in CONFL2.
E


Enter CONFL2
Initialize SLS and SLE
Count Requirements For Each Attribute Group
Call Transportation Routine
Can Assignment Be Made
Yes
Record Schedule for This Activity
RETCODE2 ← 1
Return
No
Find New Values For SLS and SLE
SLE > Window end Time
No
Yes
RETCODE2 ← 0
Return

APPENDIX B

SOURCE LISTING AND SAMPLE OUTPUT OF FINAL PROGRAM

```
STMT LEVEL NEST
  1                 SCHED5: PROC OPTIONS(MAIN);

                    /*
                      THIS PROGRAM SCHEDULES MULTIPLE RESOURCE CLASSES.  EACH RESOURCE
                      CLASS HAS ONE OR MORE ATTRIBUTES; THE RESOURCES REQUIRED BY AN
                      ACTIVITY ARE  SPECIFIED IN TERMS OF ATTRIBUTE GORUPS.

                     MAJOR PROGRAM VARIABLES:
                            ACTTBL      ACTIVITY TABLE
                            RESTBL      RESOURCE CLASS TABLE

                            RQTBL       REQUIREMENT TABLE CODED NUMERICALLY
                            STK         PUSHDOWN STACK USED TO TRAVERSE WINDOW TREE
                            PSTK        PUSHDOWN STACK USED TO GENERATE PERMUTATIONS
                            D           MATRIX USED TO REPRESENT DEPENDENCY RELATION
                                                BETWEEN PAIRS OF ACTIVITIES
                            SLVEC       VECTOR OF TENTATIVE ALLOCATION TIMES - ONE
                                                VECTOR PER RESOURCE CLASS
                            SUB         SUBSET OF ACTIVITIES BEING SCHEDULED
                            SUBRES SUBSET OF ATTRIBUTE GROUPS REQUIRED BY CURRENT ACTIVITY
                            MAXAC       MAXIMUM # OF ACTIVITIES
                            MAXRES      MAXIMUM # OF RESOURCES
                            MAXW        MAXIMUM # OF WINDOWS PER ACTIVITY
                            MAXRQ       MAXIMUM # OF REQUIREMENTS (TOTAL MAXIMUM)
                            MAXATR - MAXIMUM # OF ATTRIBUTE GROUPS
                            ACTCT       ACTUAL COUNT OF ACTIVITIES
                            RESCT       ACTUAL COUNT OF RESOURCES
                            RQCT        ACTUAL COUNT OF REQUIREMENTS
                            DCOUNT      # OF ACTIVITIES IN LARGEST SUBSET
                            LPERM - LAST SUCCESSFUL PERMUTATION
                    */
  2     1            DCL (MAXAC, MAXRES, MAXW, MAXRQ, ACTCT, RESCT, RQCT, I, J, K, MAXATR,
                          RTCODE1, RTCODE2, ROW,DCOUNT, SCT,LVL,NODE,PLVL)
                                FIXED BIN INIT(0);
                    /* READ INPUT PARAMETERS */

  3     1            GET LIST (MAXAC, MAXRES, MAXW, MAXRQ, MAXATR);
  4     1            MAXW = MIN(MAXW,8);

  5     1           BLK1: BEGIN;
  6     2            DCL  1 ACTTBL(MAXAC),
                            2 ACT# CHAR(4),
                            2 ACTNAME CHAR(8),
                            2 ACTTIME FIXED BIN, /* ACTUAL TIME REQUIRED */
                            2 ACTWINDOWS(MAXW),
                              3   ACTSTRT FIXED BIN,  /* WINDOW START TIME */
                              3   ACTEND  FIXED BIN;  /* WINDOW END TIME */
  7     2            DCL   1 RESTBL(MAXRES),
                            2 RES# CHAR(4),
                            2 RESNAME CHAR(8),
                            2 RESUNITS FIXED BIN,  /* # OF UNITS IN CLASS */
                            2 RESATR(10) FIXED BIN;
  8     2            DCL   1 RQTBLA(MAXRQ),
                            2 RQACTA CHAR(4),
                            2 RQATRA FIXED BIN;
  9     2            DCL CARDCODE CHAR(1), BUF CHAR(79);
```

```
      SCHED5: PROC OPTIONS(MAIN);                                                  SCHED 10

 STMT LEVEL NEST

  10     2              DCL ATTBL(MAXRES+1,MAXATR+1) FIXED BIN,                              SCHED0570
                             RESQTY(MAXRES) FIXED BIN;                                       SCHED0580
                                                                                             SCHED0590
                       /* READ INPUT DATA FOR ACTIVITIES, RESOURCE CLASSES AND              SCHED0600
                          REQUIREMENTS                                                       SCHED0610
                       */                                                                    SCHED0620
  11     2              ON ENDFILE(SYSIN) GO TO LAST_CARD;                                   SCHED0630
  13     2               ATTBL = 1;                                                          SCHED0640
                                                                                             SCHED0650
  14     2             READCARD:                                                             SCHED0660
                        GET EDIT (CARDCODE,BUF)(COL(1),A(1),A(79));                          SCHED0670
  15     2               IF CARDCODE = '1' THEN                                              SCHED0680
  16     2                      DO;               /* ACTIVITY TABLE INPUT */                 SCHED0690
  17     2      1                  ACTCT = ACTCT +1;                                         SCHED0700
  18     2      1                  GET STRING (BUF) EDIT(ACTTBL(ACTCT))                      SCHED0710
                                          (A(4),A(8),(17)F(4));                              SCHED0720
  19     2      1               END;                                                         SCHED0730
  20     2                 ELSE IF CARDCODE = '2' THEN                                       SCHED0740
  21     2                      DO;               /* RESOURCE CLASS TABLE INPUT */           SCHED0750
  22     2      1                  RESCT = RESCT+1;                                          SCHED0760
  23     2      1                  GET STRING (BUF) EDIT (RESTBL(RESCT))                     SCHED0770
                                          (A(4),A(8),(11) F(4));                             SCHED0780
  24     2      1                  DO I=1 TO 10 WHILE(RESATR(RESCT,I) >0);                   SCHED0790
  25     2      2                     ATTBL(RESCT,RESATR(RESCT,I)) = 0;                      SCHED0800
  26     2      2                  END;                                                      SCHED0810
  27     2      1                  RESQTY(RESCT) = RESUNITS(RESCT);                          SCHED0820
  28     2      1               END;                                                         SCHED0830
  29     2                 ELSE IF CARDCODE = '3' THEN                                       SCHED0840
  30     2                      DO;               /* REQUIREMENTS INPUT */                   SCHED0850
  31     2      1                  RQCT = RQCT+1;                                            SCHED0860
  32     2      1                  GET STRING (BUF) EDIT(RQTBLA(RQCT))(A(4),F(4));           SCHED0870
  33     2      1               END;                                                         SCHED0880
  34     2                 ELSE PUT SKIP EDIT (CARDCODE,BUF,' INVALID CARDCODE')             SCHED0890
                                          (A(1),A(79),A);                                    SCHED0900
  35     2              GO TO READCARD;                                                      SCHED0910
                                                                                             SCHED0920
  36     2             LAST_CARD:                                                            SCHED0930
                        IF ACTCT = 0 | RESCT = 0 | RQCT = 0                                  SCHED0940
  37     2                      THEN DO;                                                     SCHED0950
  38     2      1                     PUT SKIP EDIT ('MISSING INPUT DATA')(A);               SCHED0960
  39     2      1                     STOP;                                                  SCHED0970
  40     2      1                    END;                                                    SCHED0980
  41     2               ATTBL(RESCT+1,*) = 1;                                               SCHED0990
  42     2               ATTBL(*,MAXATR+1) = 0;                                              SCHE1000
                       /* PRINT TABLES */                                                    SCHE1010
                                                                                             SCHE1020
  43     2             PUT EDIT (' TABLE OF ACTIVITIES ')(PAGE,X(20),A,SKIP(1));             SCHE1030
  44     2               PUT EDIT ('ACT #', 'TIME REQ', 'WINDOWS')                           SCHE1040
                             (SKIP(1),A,COL(14),A,COL(28),A);                                SCHE1050
  45     2             PUT EDIT ( (ACTTBL(I) DO I=1 TO ACTCT))                               SCHE1060
                              (SKIP(1),X(1),A(4),X(1),A(8),X(1),F(4),                        SCHE1070
                               (MAXW)(X(6),F(4),X(1),F(4)));                                 SCHE1080
  46     2             PUT EDIT ( 'TABLE OF RESOURCE CLASSES')(SKIP(3),X(10),A);             SCHE1090
  47     2               PUT EDIT('CLASS', '# OF UNITS', 'ATTRIBUTES')                       SCHE1100
                             (SKIP(1),COL(8),A,COL(23),A,COL(37),A);                         SCHE1110
```

```
SCHED5: PROC OPTIONS(MAIN);                                              SCHED 10

STMT LEVEL NEST

 48    2          PUT EDIT ((RESTBL(I) DO I=1 TO RESCT))
                            (SKIP(1),X(10),A(4),X(1),A(8),X(1),F(4),X(7),(10)F(4));
 49    2          PUT EDIT (' TABLE OF REQUIREMENTS ')(PAGE,X(20),A,SKIP(1));
 50    2           PUT EDIT('ACTIVITY', 'ATTRIBUTE GROUP')
                         (SKIP(1),COL(6),A,COL(20),A);
 51    2          PUT EDIT ((RQTBLA(I) DO I=1 TO RQCT))
                            (SKIP(1),X(10),A(4),X(10),F(4));
                                 /*
                   PUT EDIT('ATTRIBUTE MATRIX', ((ATTBL(I,J) DO J=1 TO
                          MAXATR+1) DO I=1 TO RESCT+1))
                        (PAGE,A,SKIP(2),(RESCT+1)((MAXATR+1)(F(4)),SKIP));
                                */
 52    2          BLK2:  BEGIN;
 53    3           DCL LOOKA ENTRY RETURNS(FIXED BIN);
 54    3           DCL 1 RQTBL (RQCT),
                         2(RQACT#, RQATR#) FIXED BIN;
 55    3           DCL D(ACTCT,ACTCT) BIT(1);

                  /* LOOK UP EACH ACTIVITY & ATTRIB.  IN RQTBLA, AND PLACE THE ROW
                     POSITIONS IN THE CORRESPONDING POSITION IN RQTBL, THUS CONSTRUCT-
                     ING A NUMERICAL REQUIREMENT TABLE
                  */
 56    3           DO I=1 TO RQCT;
 57    3     1        ROW = LOOKA(RQACTA(I));
 58    3     1        IF ROW = 0 THEN GO TO TBL_ERROR;
 60    3     1        RQACT#(I) = ROW;
 61    3     1         RQATR#(I) = RQATRA(I);
 62    3     1     END;
 63    3           GO TO BUILD_D;

 64    3          TBL_ERROR:
                   PUT SKIP EDIT (RQACTA(I),             ' ITEM NOT IN TABLE')
                                              ( A(4),X(2),A(4),A);
 65    3           STOP;

                  /*  CONSTRUCT D MATRIX BY ENTERING A 1 IN D(I,J) AND D(J,I)
                      IF ACT(I) AND ACT(J) MUST SHARE AT LEAST 1 ATTRIB.  CLASS
                  */
 66    3          BUILD_D:
                  D = '0'B;
 67    3           DO I = 1 TO RQCT-1;
 68    3     1       DO J = I+1 TO RQCT;
 69    3     2          IF RQACT#(I) ¬= RQACT#(J) & RQATR#(I) = RQATR#(J)
 70    3     2             THEN DO;
 71    3     3                    D(RQACT#(I),RQACT#(J)) = '1'B;
 72    3     3                    D(RQACT#(J),RQACT#(I)) = '1'B;
 73    3     3                   END;
 74    3     2       END;
 75    3     1     END;
                  /* NOW USE WARSHALL'S ALGORITHM TO GET THE PATH MATRIX
                     CORRESPONDING TO THE ADJACENCY MATRIX D.
                  */
 76    3           DO J=1 TO ACTCT;
 77    3     1       DO I = 1 TO ACTCT;
 78    3     2          IF D(I,J) THEN  D(I,*) = D(I,*) | D(J,*);
```

```
STMT LEVEL NEST

 80    3    2        END;                                                                  SCHE1670
 81    3    1      END;                                                                    SCHE1680
 82    3           DO I = 1 TO ACTCT;                                                      SCHE1690
 83    3    1        D(I,I) = '1'B;                                                        SCHE1700
 84    3    1      END;                                                                    SCHE1710
                                                                                           SCHE1720
                  /* EACH ROW IN THE D MATRIX SPECIFIES A SUBSET OF ACTIVITIES THAT         SCHE1730
                     MUST BE SCHEDULED INTERDEPENDENTLY.                                   SCHE1740
                     FIND THE # OF ACTIVITIES IN THE LARGEST SUBSET                        SCHE1750
                  */                                                                       SCHE1760
 85    3           DCOUNT = 0;                                                             SCHE1770
 86    3           DO I = 1 TO ACTCT;                                                      SCHE1780
 87    3    1        K = 0;                                                                SCHE1790
 88    3    1        DO J = 1 TO ACTCT;                                                    SCHE1800
 89    3    2           IF D(I,J) THEN K = K+1;                                            SCHE1810
 91    3    2        END;                                                                  SCHE1820
 92    3    1        DCOUNT = MAX(DCOUNT,K);                                               SCHE1830
 93    3    1      END;                                                                    SCHE1840
 94    3          BLK3: BEGIN;                                                             SCHE1850
 95    4            DCL (A(RESCT+1),B(MAXATR+1),C1(RESCT+1,MAXATR+1),                      SCHE1860
                             X(RESCT+1,MAXATR+1)) FIXED BIN;                              SCHE1870
                                                                                           SCHE1880
 96    4           DCL (STK(DCOUNT), PSTK(DCOUNT),           SUB(DCOUNT)) FIXED BIN;       SCHE1890
 97    4           DCL 1 SLVECTORS(0:MAXATR),                                              SCHE1900
                         2 SLPT FIXED BIN,                                                 SCHE1910
                         2 SLVEC (DCOUNT),                                                 SCHE1920
                            3 (SLSTRT,SLEND) FIXED BIN;                                    SCHE1930
 98    4           DCL SUBRES(MAXATR)FIXED BIN;                                            SCHE1940
 99    4            DCL CONFL2 ENTRY(BIT(1));                                              SCHE1950
100    4            DCL LPERM(DCOUNT,DCOUNT) FIXED BIN;                                    SCHE1960
                  /* BEGIN TREE TRAVERSAL FOR SUBSETS OF ACTIVITIES THAT REQUIRE           SCHE1970
                     INTER-DEPENDENT SCHEDULING                                            SCHE1980
                  */                                                                       SCHE1990
101    4          LOOP_1:                                                                  SCHE2000
                   DO ROW = 1 TO ACTCT;                                                    SCHE2010
102    4    1        DO I = 1 TO ACTCT;                                                    SCHE2020
103    4    2           IF D(ROW,I) THEN GO TO SCH_SUBSET;                                 SCHE2030
105    4    2        END;                                                                  SCHE2040
106    4    1        GO TO END_LOOP_1;                                                     SCHE2050
107    4    1       SCH_SUBSET:                                                            SCHE2060
                                                                                           SCHE2070
                     /* IDENTIFY ACTIVITIES IN THE SUBSET SPECIFIED BY THIS ROW. IF        SCHE2080
                        D(ROW,I) = 1 PLACE ACTIVITY I INTO THE SUB VECTOR, THEN ZERO       SCHE2090
                        OUT ROW I IN THE D MATRIX SINCE ROW I WILL BE IDENTICAL TO         SCHE2100
                        THE CURRENT ROW AND WILL DEFINE THE SAME SUBSET OF ACTIVITIES.     SCHE2110
                     */                                                                    SCHE2120
                     SUB=0;                                                                SCHE2130
108    4    1        SCT=0;                                                                SCHE2140
109    4    1        PUT EDIT ('ATTEMPTING TO SCHEDULE THE FOLLOWING ACTIVITIES',          SCHE2150
                               'ACT #     TIME REQUIRED          WINDOWS')                 SCHE2160
                               (PAGE,A,SKIP(2),A);                                         SCHE2170
110    4    1        DO I=1 TO ACTCT;                                                      SCHE2180
111    4    2          IF D(ROW,I) = '1'B THEN                                             SCHE2190
112    4    2             DO;                                                              SCHE2200
113    4    3                SCT = SCT+1;                                                  SCHE2210
```

```
STMT LEVEL NEST

114    4    3              SUB(SCT)=I;
115    4    3              IF I ¬= ROW THEN D(I,*) = '0'B;
117    4    3              PUT SKIP EDIT (ACT#(I),ACTTIME(I),(ACTWINDOWS(I,J)
                                     DO J=1 TO MAXW WHILE (ACTEND(I,J) ¬= 0)))
                                (A(4),X(7),F(4),X(8),(MAXW)(F(4),X(1),F(4),
                                 X(3))));
118    4    3           END;
119    4    2        END;

                 /* BEGIN TRAVERSAL OF WINDOW TREE FOR SUBSET OF ACTIVITIES */

120    4    1        LPERM=0;
121    4    1        LVL = 1;
122    4    1     FIRST_WINDOW:
                     NODE = 1;
                     /* PLACE NEW NODE ON STACK AND CHECK FOR CONFLICT */

123    4    1     PUSH_ONTO_STACK:
                     STK(LVL) = NODE;
124    4    1        CALL CONFL1;
125    4    1        IF RTCODE1= 1 THEN
126    4    1           DO;
                           /* NO CONFLICT DETECTED; GO TO NEXT LEVEL */
127    4    2           IF LVL = SCT THEN GO TO OUTPUT_SOLUTION;
129    4    2           DO I=1 TO LVL;
130    4    3              LPERM(LVL,I) = PSTK(I);
131    4    3           END;
132    4    2           LVL = LVL + 1;
133    4    2           GO TO FIRST_WINDOW;
134    4    2        END;
                     /* CONFLICT DETECTED. CHECK NEXT WINDOW OR GO TO PREVIOUS LEVEL*/

135    4    1     NEXT_WINDOW:
                     NODE = NODE +1;
136    4    1        IF NODE <= MAXW & ACTEND(SUB(LVL),NODE) ¬= 0
137    4    1              THEN GO TO PUSH_ONTO_STACK;
138    4    1        IF LVL=1 THEN DO;
140    4    2                        PUT EDIT((60)'-',(60)'-')(SKIP(2),A,SKIP(1),A);
141    4    2                        GO TO END_LOOP_1;
142    4    2                     END;
143    4    1        LVL = LVL-1;
144    4    1        NODE = STK(LVL);
145    4    1        GO TO NEXT_WINDOW;

146    4    1     OUTPUT_SOLUTION:
                    CALL CONFL2('1'B);
147    4    1       IF RTCODE2 = 0 THEN GO TO NEXT_WINDOW;
149    4    1       PUT EDIT((60)'-')(SKIP(2),A);
150    4    1        PUT EDIT ('SCHEDULE FOR ABOVE ACTIVITIES',
                              'ACT#       WINDOW       ACTUAL')
                             (SKIP(2),X(10),A,SKIP(1),X(10),A);
151    4    1        DO I = 1 TO LVL;
152    4    2          K = SUB(PSTK(I));
153    4    2          PUT SKIP EDIT (ACT#(K),ACTWINDOWS(K,STK(PSTK(I))),
                                 SLVEC(O,I))
```

```
STMT LEVEL NEST

                              (X(9),A(4),(2)(X(3),F(4),X(1),F(4))));
154    4    2         END;
155    4    1       PUT EDIT('ASSIGNMENTS OF RESOURCE CLASSES TO ATTRIBUTE GROUPS',
                         'RESOURCE CLASS     ATTRIBUTE GROUP     # OF UNITS')
                         (SKIP(2),A,SKIP(1),A);
156    4    1       DO I=1 TO RESCT;
157    4    2          DO J=1 TO MAXATR;
158    4    3             IF X(I,J) ¬= 0 THEN
159    4    3                PUT EDIT (RES#(I),J,X(I,J))
                                (SKIP(1),COL(5),A(4),COL(22),F(4),COL(39),F(4));
160    4    3          END;
161    4    2       END;
162    4    1       PUT EDIT ('RESOURCE ASSIGNMENTS','CLASS', 'TIMES ASSIGNED')
                         (SKIP(2),COL(20),A,SKIP(1),A,COL(15),A);
163    4    1       DO I=1 TO RESCT;
164    4    2          PUT SKIP EDIT(RES#(I)) (X(1),A(4));
165    4    2          ICOL=10;
166    4    2          DO J =1 TO MAXATR;
167    4    3             IF X(I,J) ¬= 0 THEN
168    4    3                DO K=1 TO SLPT(J);
169    4    4                   PUT SKIP(0) EDIT('(',SLVEC(J,K),')')
                                     (COL(ICOL),A,F(4),X(1),F(4),A);
170    4    4                   ICOL = ICOL+12;
171    4    4                   IF ICOL>110 THEN ICOL=10;
173    4    4                END;
174    4    3          END;
175    4    2          END;
176    4    1         GO TO NEXT_WINDOW;




177    4    1       END_LOOP_1:
                     END LOOP_1;

                    /* * * * * * * * * * * * * * * * * * * * *        */

178    4            CONFL1: PROC;

                    /* GENERATE PERMUTATIONS OF WINDOWS IN THE STACK UNTIL A PERMUTA-
                       TION IS REACHED FOR WHICH A SCHEDULE CAN BE FOUND
                    */
179    5            DCL (I,J,K,L) FIXED BIN;
180    5            SLVECTORS=0;
181    5            SUBRES=0;
182    5            IF LVL = 1 THEN
183    5              DO;
184    5    1           I = SUB(1);
185    5    1           CALL SCANRQ(I);
186    5    1           PSTK(1) = 1;
187    5    1           DO J =1 TO MAXATR WHILE(SUBRES(J) > 0);
188    5    2              K = SUBRES(J);
189    5    2              SLSTRT(K,1) = ACTSTRT(I,STK(1));
190    5    2              SLEND(K,1) = SLSTRT(K,1) + ACTTIME(I);
191    5    2              SLPT(K) = 1;
```

```
SCHED5: PROC OPTIONS(MAIN);                                              SCHED 10

STMT LEVEL NEST

 192     5    2          END;
 193     5    1          RTCODE1 = 1;
 194     5    1          SLSTRT(0,1) = ACTSTRT(1,STK(1));
 195     5    1          SLEND(0,1) = SLSTRT(0,1) + ACTTIME(1);
 196     5    1             SLPT(0) = 1;
 197     5    1          RETURN;
 198     5    1        END;
                    /* BEGIN GENERATING PERMUTATIONS IN LEXICAL ORDER,STARTING WITH THE
                        PERMUTATION WHICH PRODUCED A SCHEDULE AT THE PREVIOUS LEVEL.
                        RESTORE THIS PREVIOUS PERMUTATION IN PSTK, AND CALL CONFL2 REPEAT-
                        EDLY TO RESTORE THE PREVIOUS SCHEDULE.
                    */
 199     5            PSTK=0;
 200     5            DO PLVL=1 TO LVL-1;
 201     5    1          PSTK(PLVL) = LPERM(LVL-1,PLVL);
 202     5    1          CALL CONFL2('0'B);
 203     5    1       END;
 204     5            PLVL = LVL;
 205     5            PSTK(PLVL) = LVL;
 206     5            GO TO CALL_C2;
 207     5          NEXT_LVL:
                     PSTK(PLVL) = 1;
 208     5          CHECK_CONFL2:
                     IF PLVL > 1 THEN
 209     5              DO I = 1 TO PLVL-1;
 210     5    1          IF PSTK(I) =PSTK(PLVL) THEN GO TO NEXT_NO;
 212     5    1         END;
                    /* COMPARE THIS PERMUTATION WITH THE PERMUTATION OF THE PREVIOUS LEVEL
                        AND CHECK FOR VIOLATIONS OF LEXICAL ORDERING
                    */
 213     5            K=0;
 214     5            DO I=1 TO PLVL;
 215     5    1          K = K+1;
 216     5    1          IF PSTK(K) =  LVL  THEN K = K+1;
 218     5    1          IF PSTK(K) < LPERM(LVL-1,I) THEN GO TO NEXT_NO;
 220     5    1          IF PSTK(K) > LPERM(LVL-1,I) THEN GO TO CALL_C2;
 222     5    1       END;
 223     5          CALL_C2:
                      CALL CONFL2('0'B);
 224     5           IF RTCODE2 = 0 THEN GO TO NEXT_NO;

                     /* NO CONFLICT DETECTED */
 226     5           IF PLVL = LVL THEN DO;
 228     5    1                          RTCODE1 = 1;
 229     5    1                          RETURN;
 230     5    1                       END;
 231     5           PLVL = PLVL+1;
 232     5           GO TO NEXT_LVL;

 233     5          NEXT_NO:       /* CONFLICT FOUND */
                     IF PSTK(PLVL) < LVL THEN
 234     5               DO;
 235     5    1             PSTK(PLVL) = PSTK(PLVL) + 1;
 236     5    1             GO TO CHECK_CONFL2;
 237     5    1          END;
```

```
STMT LEVEL NEST

238    5            IF PLVL = 1  THEN DO;
240    5    1                         RTCODE1 =0;
241    5    1                         RETURN;
242    5    1                      END;
243    5           PLVL = PLVL - 1;
                   /* REMOVE TENTATIVE SCHEDULE TIME FOR ACTIVITY POINTED TO
                      BY PSTK(PLVL)
                   */
244    5           CALL SCANRQ(SUB(PSTK(PLVL)));
245    5           DO I =1 TO MAXATR WHILE(SUBRES(I) > 0);
246    5    1         SLPT(SUBRES(I)) = SLPT(SUBRES(I)) - 1;
247    5    1      END;
248    5           SLPT(0) = SLPT(0) - 1;
249    5           GO TO NEXT_NO;
250    5           END CONFL1;

                  /*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  */


251    4          CONFL2: PROC(FINAL);
                  /*
                    THIS ROUTINE ATTEMPTS TO FIND A SCHEDULE FOR THE ACTIVITIES
                    POINTED TO BY PSTK.  PLVL IS THE # OF ACTIVITIES TO BE SCHEDULED.
                    IF PLVL > 1 THEN PLVL-1 ACTIVITIES HAVE ALREADY BEEN SCHEDULED.
                     IF FINAL = 1 THEN A FULL PREMUTATIONHAS BEEN FOUND WHICH HAS
                     THUS FAR PRODUCED NO CONFLICT.  IN THIS CASE THE ROUTINE IS USED
                     TO FIND THE ACTUAL RESOURCE ALLOCATION, IF IT CAN BE FOUND.
                  */
252    5            DCL FINAL BIT(1);
253    5          DCL (I,J,K,SLS,SLE,NEXTSLS,IPOINT,MIND,TEMP,COUNT,INF,DELT,COST,
                       WINDEND, HOLDK)
                     FIXED BIN;
254    5          DCL(C(DCOUNT),D(DCOUNT)) FIXED BIN;
255    5          DCL OVP(DCOUNT) BIT(1);
256    5          DCL TRANSP1 ENTRY(FIXED BIN, FIXED BIN,,,,,, );
257    5             INF = 32767;
258    5           IF FINAL THEN
259    5            DO;
260    5    1         SLS=0;
261    5    1         DELT = 32767;
262    5    1         SLE,WINDEND = SLS + DELT;
263    5    1       END;
264    5           ELSE
264    5            DO;
265    5    1        I = SUB(PSTK(PLVL));
266    5    1        J = STK(PSTK(PLVL));
267    5    1        CALL SCANRQ(I);

                  /* SET TENTATIVE START TIME = START TIME OF WINDOW */
268    5    1        SLS = ACTSTRT(I,J);
269    5    1        DELT = ACTTIME(I);
270    5    1        SLE = SLS + DELT;
271    5    1        WINDEND = ACTEND(I,J);
272    5    1       END;
```

```
STMT LEVEL NEST

 273     5           COUNT_REQ:
                        NEXTSLS = WINDEND;

                     /* FOR EACH ATTRIBUTE GROUP I, SET B(J) = TO THE # OF UNITS
                        REQUIRED DURING <SLS,SLE>. INITIALIZE B(I) TO 1 IF THE CURRENT
                        ACTIVITY REQUIRES THE ATTRIBUTE I.         */
 274     5              B = 0;
 275     5             IF ¬FINAL THEN
 276     5              DO I = 1 TO MAXATR WHILE (SUBRES(I) > 0);
 277     5      1          B(SUBRES(I)) = 1;
 278     5      1       END;
 279     5              DO I = 1 TO MAXATR;
 280     5      1         KOUNT = 0;
 281     5      1         IF SLPT(I) = 0 THEN GO TO BYPASS_COUNT;
 283     5      1         DO J = 1 TO SLPT(I);
 284     5      2            C(J) = SLSTRT(I,J);
 285     5      2            D(J) = SLEND(I,J);
 286     5      2            OVP(J) = '0'B;
 287     5      2         END;

                     /*  FOR THE ATTRIBUTE I, C & D CONTAIN START & END TIMES OF ACTI-
                         VITIES ALREADY SCHEDULED.  ORDER THESE TIMES BY INCREASING
                         ORDER OF START TIME
                     */
 288     5      1         IF SLPT(I) > 1 THEN
 289     5      1            DO J=1 TO SLPT(I) -1;
 290     5      2              IF C(J) > C(J+1) THEN
 291     5      2                  DO K = J+1 BY -1 TO 2 WHILE (C(K)<C(K-1));
 292     5      3                     TEMP = C(K);
 293     5      3                     C(K) = C(K-1);
 294     5      3                     C(K-1) = TEMP;
 295     5      3                     TEMP = D(K);
 296     5      3                     D(K) = D(K-1);
 297     5      3                     D(K-1) = TEMP;
 298     5      3                  END;
 299     5      2            END;

                     /* DETERMINE THE EARLIEST TIME (AFTER SLS) THAT A UNIT MIGHT
                        BECOME AVAILABLE
                     */
 300     5      1         DO J = 1 TO SLPT(I);
 301     5      2            IF D(J) > SLS & D(J) < NEXTSLS THEN NEXTSLS = D(J);
 303     5      2         END;

                     /* FIND VALUE FOR IPOINT SUCH THAT C(IPOINT) <= SLS &
                        C(IPOINT+1) >= SLS
                     */
 304     5      1         IF  SLS<= C(1) THEN IPOINT = 0;
 306     5      1            ELSE IF SLS >= C(SLPT(I)) THEN IPOINT = SLPT(I);
 308     5      1            ELSE DO J = 1 TO SLPT(I) WHILE (C(J) < SLS);
 309     5      2                    IPOINT = J;
 310     5      2                 END;

                     /* COUNT ACTIVITIES STARTING BEFORE SLS & ENDING AFTER SLS  */
 311     5      1         IF IPOINT > 0 THEN
```

SCHED5: PROC OPTIONS(MAIN); SCHED 10

```
STMT LEVEL NEST

312    5    1             DO J = 1 TO IPOINT;
313    5    2               IF D(J) > SLS THEN
314    5    2                  DO;
315    5    3                    OVP(J) = '1'B;
316    5    3                    KOUNT = KOUNT+1;
317    5    3                  END;
318    5    2             END;
                  /* IDENTIFY ACTIVITIES THAT START DURING <SLS,SLE>.
                     FOR EACH SUCH ACTIVITY, SEE WHETHER IT CAN BE MATCHED WITH AN
                     EARLIER OVERLAPPING ACTIVITY.
                  */
319    5    1          IF IPOINT < SLPT(I) THEN
320    5    1             DO J=IPOINT+1 TO SLPT(I);
321    5    2               IF C(J) < SLE THEN
322    5    2                 DO;
323    5    3                   MIND = C(J);
324    5    3                   HOLDK = 0;
325    5    3                   IF J > 1 THEN
326    5    3                     DO K=1 TO J-1;
327    5    4                        IF D(K) <=MIND & OVP(K) = '1'B THEN
328    5    4                          DO;
329    5    5                             MIND = D(K);
330    5    5                             HOLDK = K;
331    5    5                          END;
332    5    4                     END;
333    5    3                   IF HOLDK > 0 THEN D(HOLDK) = D(J);
335    5    3                     ELSE DO;
336    5    4                            OVP(J) = '1'B;
337    5    4                            KOUNT = KOUNT+1;
338    5    4                          END;
339    5    3                 END;
340    5    2             END;
341    5    1       BYPASS_COUNT:
                         B(I) = B(I) + KOUNT;
342    5    1        END;
                  /* PREPARE TO CALL TRANSPORTATION ROUTINE   */
343    5             DO I=1 TO RESCT+1;
344    5    1           A(I) = RESQTY(I);
345    5    1          C1(I,*) = ATTBL(I,*);
346    5    1        END;
347    5             X=0;
348    5             TEMP = SUM(A) - SUM(B);
349    5             IF TEMP >= 0
350    5                  THEN DO;
351    5    1                     B(MAXATR+1) = TEMP;
352    5    1                     A(RESCT+1) = 0;
353    5    1                   END;
354    5                  ELSE DO;
355    5    1                     A(RESCT+1) = -TEMP;
356    5    1                     B(MAXATR+1) = 0;
357    5    1                   END;
358    5             CALL TRANSP1(RESCT+1,MAXATR+1,INF,C1,A,B,X,COST);
359    5             IF COST > 0 THEN GO TO REDUCE_REQ;

                   /* ENTER SLS,SLE IN SCHEDULE FOR EACH ATTRIBUTE GROUP REQUIRED
```

```
STMT LEVEL NEST

                            BY THIS ACTIVITY
                          */
 361    5                  IF FINAL THEN DO;
 363    5     1                          RTCODE2 = 1;
 364    5     1                          RETURN;
 365    5     1                        END;
 366    5                   DO I =1 TO MAXATR WHILE (SUBRES(I) > 0);
 367    5     1                K= SUBRES(I);
 368    5     1                SLPT(K) = SLPT(K)+1;
 369    5     1                SLSTRT(K,SLPT(K)) = SLS;
 370    5     1                SLEND(K,SLPT(K)) = SLE;
 371    5     1             END;
 372    5                   SLPT(0) = SLPT(0)+1;
 373    5                   SLSTRT(0,SLPT(0)) = SLS;
 374    5                   SLEND(0,SLPT(0)) = SLE;
 375    5                   RTCODE2 = 1;
 376    5                   RETURN;

 377    5                 REDUCE_REQ:

 378    5                   IF FINAL THEN DO;
 379    5     1                           RTCODE2 = 0;
 380    5     1                           RETURN;
 381    5     1                         END;
                          /* TRY NEW VALUES FOR SLS & SLE */
 382    5                   SLS = NEXTSLS;
 383    5                   SLE = SLS + DELT;
 384    5                   IF SLE > WINDEND THEN
 385    5                        DO;
 386    5     1                    RTCODE2 = 0;
 387    5     1                    RETURN;
 388    5     1                  END;
 389    5                   ELSE GO TO COUNT_REQ;

 390    5                 TRANSP1: PROC (M,N,INF,C,A,B,X,KW);
                          /*
                           ALGORITHM 293 - COLLECTED ALGORITHMS FROM CACM
                          */
 391    6                  DCL (M,N,INF,KW,A(*),B(*),C(*,*),X(*,*) )
                                 FIXED BIN;
 392    6                  DCL (I,J,U,V,K,L,S,T,GD,H,P,CIJ,XIJ,AI,BJ,LSVJ,NLVI)
                                 FIXED BIN;
 393    6                  DCL ZG BIT(1);
 394    6                  DCL (G(M),LISTU(M),NLV(M),R(N),LISTV(N),LS(0:M+N-1),
                                 NL(M+N),LSV(0:N))
                                 FIXED BIN;

 395    6                        IN: PROC;
 396    7                           LSVJ = LSV(J);
 397    7                           DO T = LSV(N) BY -1 TO LSVJ;
 398    7     1                        LS (T+1) = LS (T);
 399    7     1                     END;
 400    7                           DO  T = J TO N;
 401    7     1                        LSV(T) = LSV(T) + 1;
 402    7     1                     END;
```

```
        SCHED5: PROC OPTIONS(MAIN);                                              SCHED 10

STMT LEVEL NEST

403     7                       LS(LSVJ+1) = I;                                               SCHE6070
404     7                    END IN;                                                          SCHE6080
                                                                                              SCHE6090
405     6                    OUT: PROC;                                                       SCHE6100
406     7                       LSVJ = LSV(J);                                                SCHE6110
407     7                       DO T = LSV(J-1)+1 TO LSVJ;                                    SCHE6120
408     7      1                   IF LS(T) = I THEN DO;                                      SCHE6130
410     7      2                                                S = T;                        SCHE6140
411     7      2                                                GO TO EX;                     SCHE6150
412     7      2                                             END;                             SCHE6150
413     7      1                END;                                                          SCHE6170
414     7                     EX:                                                             SCHE6180
                                DO T = J TO N;                                                SCHE6190
415     7      1                   LSV(T) = LSV(T)-1;                                         SCHE6200
416     7      1                END;                                                          SCHE6210
417     7                       LSVJ = LSV(N);                                                SCHE6220
418     7                       DO T = S TO LSVJ;                                             SCHE6230
419     7      1                   LS(T) = LS(T+1);                                           SCHE6240
420     7      1                END;                                                          SCHE6250
421     7                    END OUT;                                                         SCHE6260
422     6                 X = 0;                                                              SCHE6270
423     6                 DO I = 1 TO M;                                                      SCHE6280
424     6      1             NLV(I) = (I-1)*N;                                                SCHE6290
425     6      1          END;                                                                SCHE6300
426     6                 LSV = 0;                                                            SCHE6310
427     6                 LISTV = 0;                                                          SCHE6320
428     6                 KW,GD = 0;                                                          SCHE6330
429     6                 DO I = 1 TO M;                                                      SCHE6340
430     6      1             H = INF;                                                         SCHE6350
431     6      1             DO J = 1 TO N;                                                   SCHE6360
432     6      2                IF C(I,J) < H THEN H = C(I,J);                                SCHE6370
434     6      2             END;                                                             SCHE6380
435     6      1             DO J = 1 TO N;                                                   SCHE6390
436     6      2                CIJ, C(I,J) = C(I,J) - H;                                     SCHE6400
437     6      2                IF CIJ = 0 THEN                                               SCHE6410
438     6      2                   DO;                                                        SCHE6420
439     6      3                      LISTV(J) = 0;                                           SCHE6430
440     6      3                      NLVI, NLV(I) = NLV(I) +1;                               SCHE6440
441     6      3                      NL(NLVI) = J;                                           SCHE6450
442     6      3                   END;                                                       SCHE6460
443     6      2             END;                                                             SCHE6470
444     6      1             KW = H*A(I)+KW;                                                  SCHE6480
445     6      1          END;                                                                SCHE6490
446     6                 DO J=1 TO N;                                                        SCHE6500
447     6      1            IF LISTV(J) = 0 THEN GO TO NEXTJ1;                                SCHE6510
449     6      1            H = INF;                                                          SCHE6520
450     6      1            DO I = 1 TO M;                                                    SCHE6530
451     6      2              IF C(I,J) = H THEN H = C(I,J);                                  SCHE6540
453     6      2            END;                                                              SCHE6550
454     6      1            DO I = 1 TO M;                                                    SCHE6560
455     6      2                CIJ, C(I,J) = C(I,J) - H;                                     SCHE6570
456     6      2                IF CIJ = 0 THEN                                               SCHE6580
457     6      2                   DO;                                                        SCHE6590
458     6      3                      NLVI,NLV(I) = NLV(I)+1;                                 SCHE6600
459     6      3                      NL(NLVI) = J;                                           SCHE6610
```

```
        SCHED5: PROC OPTIONS(MAIN);                                          SCHED 10

STMT LEVEL NEST

 460    6    3                  END;                                          SCHE6620
 461    6    2          END;                                                  SCHE6630
 462    6    1          KW = H*B(J)+KW;                                       SCHE6640
 463    6    1        NEXTJ1:                                                 SCHE6650
                      END;                                                    SCHE6660
                                                                              SCHE6670
 464    6            S2:                                                      SCHE6680
                      DO I = 1 TO M;                                          SCHE6690
 465    6    1          AI = A(I);                                            SCHE6700
 466    6    1          NLVI = NLV(I);                                        SCHE6710
 467    6    1          DO U = (I-1)*N+1 TO NLVI;                             SCHE6720
 468    6    2            IF AI = 0 THEN GO TO NEXTI2;                        SCHE6730
 470    6    2            J = NL(U);                                          SCHE6740
 471    6    2            BJ = B(J);                                          SCHE6750
 472    6    2            IF BJ = 0 THEN GO TO NEXTJ4;                        SCHE6750
 474    6    2            H,X(I,J) = MIN(AI,BJ);                              SCHE6770
 475    6    2            AI = AI-H;                                          SCHE6780
 476    6    2            B(J) = BJ - H;                                      SCHE6790
 477    6    2            CALL IN;                                            SCHE6800
 478    6    2          NEXTJ4:                                               SCHE6810
                         END;                                                 SCHE6820
                   /* BEGIN PAGE 2 */                                         SCHE6830
 479    6    1          NEXTI2:                                               SCHE6840
                         A(I) = AI;                                           SCHE6850
 480    6    1          GD = GD +AI;                                          SCHE6860
 481    6    1        END;                                                    SCHE6870
 482    6            S31:                                                     SCHE6880
 483    6              IF GD = 0 THEN GO TO S6;                               SCHE6890
 484    6            S32:                                                     SCHE6900
                       R = 0;                                                 SCHE6910
 485    6              K = 0;                                                 SCHE6920
 486    6              DO I = 1 TO M;                                         SCHE6930
 487    6    1           IF A(I) ¬= 0 THEN                                    SCHE6940
 488    6    1              DO;                                               SCHE6950
 489    6    2                K = K+1;                                        SCHE6960
 490    6    2                LISTU(K) = I;                                   SCHE6970
 491    6    2                G(I) = INF;                                     SCHE6980
 492    6    2              END;                                              SCHE6990
 493    6    1           ELSE G(I) = 0;                                       SCHE7000
 494    6    1         END;                                                   SCHE7010
 495    6            S33:                                                     SCHE7020
                       L = 0;                                                 SCHE7030
 496    6              DO U = 1 TO K;                                         SCHE7040
 497    6    1           I = LISTU(U);                                        SCHE7050
 498    6    1           NLVI = NLV(I);                                       SCHE7060
 499    6    1           DO S = (I-1)*N+1 TO NLVI;                            SCHE7070
 500    6    2             J = NL(S);                                         SCHE7080
 501    6    2             IF R(J) ¬= 0 THEN GO TO NEXTJ5;                    SCHE7090
 503    6    2             R(J) = I;                                          SCHE7100
 504    6    2             L = L+1;                                           SCHE7110
 505    6    2             LISTV(L) = J;                                      SCHE7120
 506    6    2             IF B(J) > 0 THEN GO TO S4;                         SCHE7130
 508    6    2           NEXTJ5:                                              SCHE7140
                         END;                                                 SCHE7150
 509    6    1         END;                                                   SCHE7150
```

```
STMT LEVEL NEST

510     6             IF L= 0 THEN GO TO S5;
512     6             K=0;
513     6             DO V = 1 TO L;
514     6      1         J = LISTV(V);
515     6      1         LSVJ = LSV(J);
516     6      1         DO S = LSV(J-1)+1 TO LSVJ;
517     6      2            I = LS(S);
518     6      2            IF G(I) = 0 THEN
519     6      2               DO;
520     6      3                  G(I) = J;
521     6      3                  K = K+1;
522     6      3                  LISTU(K) = I;
523     6      3               END;
524     6      2         END;
525     6      1      END;
526     6             IF K=0 THEN GO TO S5;
528     6             GO TO S33;

529     6          S4:
                      H = B(J);
530     6             P = J;

               /* BEGIN PAGE 2 COLUMN 2  */

531     6          MARK:
                      I = R(J);
532     6             J = G(I);
533     6             IF J = INF THEN
534     6               DO;
535     6      1          IF A(I) < H THEN H = A(I);
537     6      1          GO TO RE;
538     6      1        END;
539     6             IF X(I,J) < H THEN H = X(I,J);
541     6             GO TO MARK;
542     6          RE:
                      J =P;
543     6             B(J) = B(J) - H;
544     6             A(I) = A(I) - H;
545     6             GD = GD - H;
546     6          RE1:
                      I = R(J);
547     6             XIJ = X(I,J);
548     6             X(I,J) = XIJ +H;
549     6             IF XIJ = 0 THEN CALL IN;
551     6             J = G(I);
552     6             IF J=INF THEN GO TO S31;
554     6             XIJ, X(I,J) = X(I,J)-H;
555     6             IF XIJ = 0 THEN CALL OUT;
557     6             GO TO RE1;
558     6          S5:
                      K=0;
559     6             L=N+1;
560     6             DO J= 1 TO N;
561     6      1         IF R(J) = 0 THEN
562     6      1           DO;
```

```
SCHED5: PROC OPTIONS(MAIN);                                          SCHED 10

STMT LEVEL NEST

 563     6    2               K=K+1;                                            SCHE7720
 564     6    2               LISTV(K) = J;                                     SCHE7730
 565     6    2             END;                                                SCHE7740
 566     6    1           ELSE DO;                                              SCHE7750
 567     6    2                 L = L-1;                                        SCHE7760
 568     6    2                 LISTV(L)= J;                                    SCHE7770
 569     6    2               END;                                              SCHE7780
 570     6    1        END;                                                     SCHE7790
 571     6             H = INF;                                                 SCHE7800
 572     6             DO I= 1 TO M;                                            SCHE7810
 573     6    1          IF G(I) = 0 THEN GO TO NEXTI6;                         SCHE7820
 575     6    1          DO S = 1 TO K;                                         SCHE7830
 576     6    2             J = LISTV(S);                                       SCHE7840
 577     6    2             IF C(I,J) < H THEN H = C(I,J);                      SCHE7850
 579     6    2          END;                                                   SCHE7860
 580     6    1        NEXTI6:                                                  SCHE7870
                       END;                                                     SCHE7880
 581     6             DO I = 1 TO M;                                           SCHE7890
 582     6    1          ZG = (G(I) ¬= 0);                                      SCHE7900
 583     6    1          NLVI = (I-1)*N;                                        SCHE7910
 584     6    1          DO S = L TO N;                                         SCHE7920
 585     6    2             J = LISTV(S);                                       SCHE7930
 586     6    2             IF ZG THEN CIJ = C(I,J);                            SCHE7940
 588     6    2               ELSE CIJ, C(I,J) = C(I,J) + H;                    SCHE7950
 589     6    2             IF CIJ = 0 THEN                                     SCHE7960
 590     6    2                DO;                                              SCHE7970
 591     6    3                   NLVI = NLVI+1;                                SCHE7980
 592     6    3                   NL(NLVI) = J;                                 SCHE7990
 593     6    3                END;                                             SCHE8000
 594     6    2          END;                                                   SCHE8010
 595     6    1          DO S = 1 TO K;                                         SCHE8020
 596     6    2             J = LISTV(S);                                       SCHE8030
 597     6    2             IF ZG THEN CIJ,C(I,J) = C(I,J)-H;                   SCHE8040
 599     6    2               ELSE CIJ = C(I,J);                                SCHE8050
 600     6    2             IF CIJ = 0 THEN                                     SCHE8060
 601     6    2               DO;                                               SCHE8070
 602     6    3                 NLVI = NLVI+1;                                  SCHE8080
 603     6    3                 NL(NLVI) = J;                                   SCHE8090
 604     6    3               END;                                              SCHE8100
 605     6    2          END;                                                   SCHE8110
 606     6    1          NLV(I) = NLVI;                                         SCHE8120
 607     6    1        END;                                                     SCHE8130
 608     6             KW = KW + H*GD;                                          SCHE8140
 609     6             GO TO S32;                                               SCHE8150
 610     6          S6: RETURN;                                                 SCHE8160
 611     6             END TRANSP1;                                             SCHE8170
 612     5             END CONFL2;                                              SCHE8180
                                                                                SCHE8190
                  /* * * * * * * * * * * * * * * * * * * * * */                 SCHE8200
 613     4        SCANRQ: PROC(I);                                              SCHE8210
                                                                                SCHE8220
                  /* SCAN RQTBL TO IDENTIFY ALL ATTRIB. S REQUIRED BY ACTIVITY I.  SCHE8230
                     PLACE THE NUMBERS OF THE GROUPS    IN SUBRES VECTOR        SCHE8240
                  */                                                            SCHE8250
 614     5         DCL(I,J,K,L,M) FIXED BIN;                                    SCHE8260
```

```
STMT LEVEL NEST

 615     5            K=0;
 616     5            SUBRES = 0;
 617     5            DO J = 1 TO RQCT;
 618     5     1        IF RQACT#(J) = I THEN DO;
 620     5     2                                  K=K+1;
 621     5     2                                  SUBRES(K) = RQATR#(J);
 622     5     2                                END;
 623     5     1      END;
 624     5            END SCANRQ;

 625     4            END BLK3;

                     /* * * * * * * * * * * * * * * * * * * * * * */
 626     3           LOOKA: PROC(ARG ) RETURNS (FIXED BIN);
 627     4            DCL (I,J,K,L,M) FIXED BIN,  ARG CHAR(4);
 628     4            DO I = 1 TO ACTCT;
 629     4     1        IF ARG = ACT#(I) THEN RETURN(I);
 631     4     1      END;
 632     4            RETURN(0);
 633     4            END LOOKA;


 634     3            END BLK2;
 635     2            END BLK1;
 636     1            END SCHED5;
```

TABLE OF ACTIVITIES

ACT #	TIME REQ	WINDOWS							
A1	1	1	3	0	0	0	0		
A2	1	1	4	7	9	0	0		
A3	3	2	5	0	0	0	0		
A4	1	1	9	0	0	0	0		
A5	3	4	7	0	0	0	0		
A6	4	5	10	0	0	0	0		
A7	2	9	11	0	0	0	0		
A8	1	10	12	13	14	0	0		
A9	2	10	14	16	18	19	21		

TABLE OF RESOURCE CLASSES

CLASS	# OF UNITS	ATTRIBUTES									
R1	4	1	3	8	0	0	0	0	0	0	0
R2	3	2	4	0	0	0	0	0	0	0	0
R3	5	5	6	7	0	0	0	0	0	0	0
R4	2	3	4	0	0	0	0	0	0	0	0
R5	6	2	7	8	0	0	0	0	0	0	0

TABLE OF REQUIREMENTS

ACTIVITY	ATTRIBUTE GROUP
A1	1
A1	5
A2	3
A2	4
A3	2
A3	5
A3	6
A4	7
A4	8
A5	2
A5	6
A6	4
A7	1
A8	4
A8	7
A9	1

```
ATTEMPTING TO SCHEDULE THE FOLLOWING ACTIVITIES

ACT#     TIME REQUIRED          WINDOWS
A1            1              1    3
A3            3              2    5
A5            3              4    7
A7            2              9   11
A9            2             10   14     16    18     19   21

-----------------------------------------------------------

          SCHEDULE FOR ABOVE ACTIVITIES
           ACT#      WINDOW      ACTUAL
          A1         1    3       1    2
          A3         2    5       2    5
          A5         4    7       4    7
          A7         9   11       9   11
          A9        10   14      10   12

ASSIGNMENTS OF RESOURCE CLASSES TO ATTRIBUTE GROUPS
RESOURCE CLASS       ATTRIBUTE GROUP      # OF UNITS
    R1                     1                  2
    R2                     2                  2
    R3                     5                  1
    R3                     6                  2

                     RESOURCE ASSIGNMENTS
CLASS            TIMES ASSIGNED
 R1        (   1    2) (   9    11) (   10    12)
 R2        (   2    5) (   4    7)
 R3        (   1    2) (   2    5) (   2    5) (   4    7)
 R4
 R5

-----------------------------------------------------------

          SCHEDULE FOR ABOVE ACTIVITIES
           ACT#      WINDOW      ACTUAL
          A1         1    3       1    2
          A3         2    5       2    5
          A5         4    7       4    7
          A7         9   11       9   11
          A9        16   18      16   18

ASSIGNMENTS OF RESOURCE CLASSES TO ATTRIBUTE GROUPS
RESOURCE CLASS       ATTRIBUTE GROUP      # OF UNITS
    R1                     1                  1
    R2                     2                  2
    R3                     5                  1
    R3                     6                  2

                     RESOURCE ASSIGNMENTS
CLASS            TIMES ASSIGNED
 R1        (   1    2) (   9    11) (   16    18)
 R2        (   2    5) (   4    7)
 R3        (   1    2) (   2    5) (   2    5) (   4    7)
 R4
 R5

-----------------------------------------------------------
```

```
          SCHEDULE FOR ABOVE ACTIVITIES
           ACT#      WINDOW      ACTUAL
          A1          1    3      1    2
          A3          2    5      2    5
          A5          4    7      4    7
          A7          9   11      9   11
          A9         19   21     19   21

ASSIGNMENTS OF RESCURCE CLASSES TO ATTRIBUTE GROUPS
RESOURCE CLASS      ATTRIBUTE GROUP      # OF UNITS
    R1                     1                 1
    R2                     2                 2
    R3                     5                 1
    R3                     6                 2

                      RESOURCE ASSIGNMENTS
CLASS           TIMES ASSIGNED
 R1       (   1     2) (    9    11) (  19    21)
 R2       (   2     5) (    4     7)
 R3       (   1     2) (    2     5) (    2     5) (    4     7)
 R4
 R5

------------------------------------------------------------
------------------------------------------------------------
```

```
ATTEMPTING TO SCHEDULE THE FOLLOWING ACTIVITIES

ACT#     TIME REQUIRED          WINDOWS
A2             1            1     4       7     9
A4             1            1     9
A6             4            5    10
A8             1           10    12      13    14

------------------------------------------------------------

           SCHEDULE FOR ABOVE ACTIVITIES
           ACT#      WINDOW      ACTUAL
           A2        1    4      1    2
           A4        1    9      1    2
           A6        5   10      5    9
           A8       10   12     10   11

ASSIGNMENTS OF RESOURCE CLASSES TO ATTRIBUTE GROUPS
RESOURCE CLASS      ATTRIBUTE GROUP       # OF UNITS
    R1                     3                  1
    R1                     8                  1
    R2                     4                  1
    R3                     7                  1

                    RESOURCE ASSIGNMENTS
CLASS          TIMES ASSIGNED
 R1       (   1     2) (   1     2)
 R2       (   1     2) (   5     9) (  10     11)
 R3       (   1     2) (  10    11)
 R4
 R5

------------------------------------------------------------

           SCHEDULE FOR ABOVE ACTIVITIES
           ACT#      WINDOW      ACTUAL
           A2        1    4      1    2
           A4        1    9      1    2
           A6        5   10      5    9
           A8       13   14     13   14

ASSIGNMENTS OF RESOURCE CLASSES TO ATTRIBUTE GROUPS
RESOURCE CLASS      ATTRIBUTE GROUP       # OF UNITS
    R1                     3                  1
    R1                     8                  1
    R2                     4                  1
    R3                     7                  1

                    RESOURCE ASSIGNMENTS
CLASS          TIMES ASSIGNED
 R1       (   1     2) (   1     2)
 R2       (   1     2) (   5     9) (  13     14)
 R3       (   1     2) (  13    14)
 R4
 R5

------------------------------------------------------------
```

```
          SCHEDULE FOR ABOVE ACTIVITIES
          ACT#      WINDOW     ACTUAL



         A2         7    9       7    8
         A4         1    9       1    2
         A6         5   10       5    9
         A8        10   12      10   11

ASSIGNMENTS OF RESOURCE CLASSES TO ATTRIBUTE GROUPS
RESOURCE CLASS      ATTRIBUTE GROUP      # OF UNITS
    R1                      3                1
    R1                      8                1
    R2                      4                2
    R3                      7                1

                    RESOURCE ASSIGNMENTS
CLASS          TIMES ASSIGNED
 R1       (   7     8) (   1     2)
 R2       (   7     8) (   5     9) (  10    11)
 R3       (   1     2) (  10    11)
 R4
 R5
----------------------------------------------------------

          SCHEDULE FOR ABOVE ACTIVITIES
          ACT#      WINDOW     ACTUAL
         A2         7    9       7    8
         A4         1    9       1    2
         A6         5   10       5    9
         A8        13   14      13   14

ASSIGNMENTS OF RESOURCE CLASSES TO ATTRIBUTE GROUPS
RESOURCE CLASS      ATTRIBUTE GROUP      # OF UNITS
    R1                      3                1
    R1                      8                1
    R2                      4                2
    R3                      7                1

                    RESOURCE ASSIGNMENTS
CLASS          TIMES ASSIGNED
 R1       (   7     8) (   1     2)
 R2       (   7     8) (   5     9) (  13    14)
 R3       (   1     2) (  13    14)
 R4
 R5
----------------------------------------------------------
----------------------------------------------------------
```

APPENDIX C

GLOSSARY OF TERMS

activity - a non-recurring event that extends over a continuous time interval and requires the use of one or more resources.

attribute group - group of all resource classes which possess the same attribute.

breadth-first search - a method of tree searching in which all nodes of a given level are processed in the same step, producing the effect of traversing all paths of the tree in parallel.

constraint - a restriction or limitation which must be taken into account when scheduling an activity.

dense solution - a solution to a scheduling problem which minimizes the total elapsed time between the starting time of the first activity and the ending time of the last activity.

depth-first search - a method of tree searching in which all paths are examined in series.

distributed solution - a solution to a scheduling problem which imposes a uniform distribution of activity assignments over a period of time.

earliest schedule - a solution to a scheduling problem in which the last activity is completed as early as possible.

ending time - the time at which an activity will complete the utilization of resources allocated to it.

resource assignment - allocation of a resource unit to an activity for a specified time interval.

resource class - a collection of identical resource units.

resource unit - a person or a reusable item.

starting time - the time at which an activity will begin utilization of resources allocated to it.

tree structured search - a search for a solution to a problem which is performed by examining alternatives in a manner corresponding to the traversal of a tree.

uniformly distributed utilization - allocation of resource units in such a way that all units within a given class are allocated for approximately equal lengths of time.

window - an interval of time during which an activity may be scheduled.

VITA

Martin James Wertheim

Candidate for the Degree of

Master of Science

Thesis: TREE STRUCTURED ALGORITHMS FOR SCHEDULING ACTIVITIES AND RESOURCES IN A CONTINUUM OF TIME

Major Field: Computing and Information Science

Biographical:

Personal Data: Born in Rochester, New York, July 27, 1947, the son of William and Helen Wertheim.

Education: Graduated from Benjamin Franklin High School, Rochester, New York, in June, 1965; received Bachelor of Science degree in Mathematics from Duke University in 1969; completed requirements for the Master of Science degree at Oklahoma State University in July, 1973.

Professional Experience: Programmer/Analyst, Texas Instruments, Incorporated, 1969-1971; Graduate Assistant, Oklahoma State University, 1972-1973.